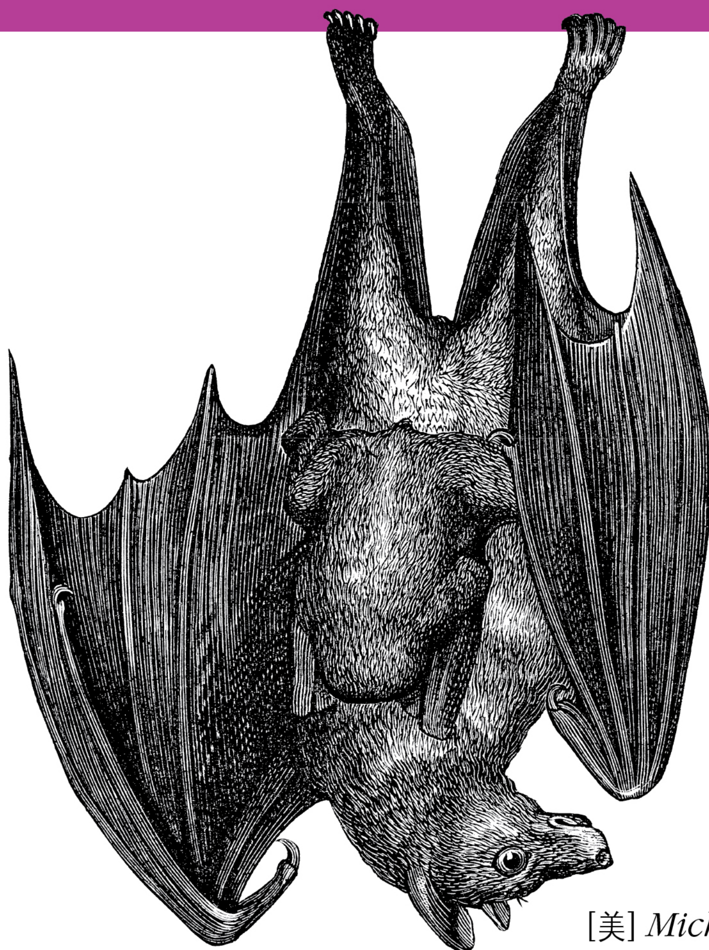
图灵程序设计丛书

Introducing Regular Expressions

学习正则表达式



[美] *Michael Fitzgerald* 著

王热宇 译

O'REILLY®

人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明

 图灵程序设计丛书

学习正则表达式

Introducing Regular Expressions

[美] Michael Fitzgerald 著
王热宇 译

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

O'Reilly Media, Inc.授权人民邮电出版社出版

人 民 邮 电 出 版 社
北 京

图书在版编目（C I P）数据

学习正则表达式 / (美) 菲茨杰拉德
(Fitzgerald,M.) 著 ; 王热宇译. -- 北京 : 人民邮电
出版社, 2013.4
（图灵程序设计丛书）
书名原文: Introducing regular expressions
ISBN 978-7-115-31149-8

Ⅰ. ①学… Ⅱ. ①菲… ②王… Ⅲ. ①正则表达式
Ⅳ. ①TP301.2

中国版本图书馆CIP数据核字(2013)第041267号

内 容 提 要

本书从正则表达式的基本概念讲起，到编写完整的 sed 和 Perl 脚本，再到转换 HTML 文件，将这种强大的工具解释得清晰透彻。书中贯穿了大量简洁明了的示例，旨在让读者轻松掌握正则表达式。此外，书中各在线和桌面工具一应俱全，并介绍了进阶参考资料，是一本不可多得的正则表达式入门好书。

本书适合对正则表达式感兴趣的程序员和互联网从业者。

图灵程序设计丛书
学习正则表达式

◆ 著 [美] Michael Fitzgerald
译 王热宇
责任编辑 朱 巍
◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京 印刷
◆ 开本：800×1000 1/16
印张：8.75
字数：167千字 2013年 4 月第 1 版
印数：1 – 3 500册 2013年 4 月北京第 1次印刷
著作权合同登记号 图字：01-2012-8416号
ISBN 978-7-115-31149-8

定价：35.00元
读者服务热线：(010)51095186转604 印装质量热线：(010)67129223
反盗版热线：(010)67171154

版权声明

O’Reilly Media, Inc.介绍

O’Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O’Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O’Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O’Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 Make 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O’Reilly 的会议和峰会聚集了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O’Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版、在线服务或者面授课程，每一项 O’Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O’Reilly Radar 博客有口皆碑。”

——Wired

“O’Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——Business 2.0

“O’Reilly Conference 是聚集关键思想领袖的绝对典范。”

——CRN

“一本 O’Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去，Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——Linux Journal

目录

前言

本书通过示例介绍如何编写正则表达式，旨在让读者轻松掌握正则表达式。事实上，笔者几乎将所涉及的每一个概念都通过示例展示了出来，读者很容易模仿尝试。

正则表达式有助于找到文本字符串中的各种模式。更确切地说，正则表达式是经过专门编写的文本字符串，用来匹配字符串（尤其是文件内字符串）集合中符合该模式的所有字符串。

正则表达式最早出现于美国数学家斯蒂芬·克莱尼编写的 *Introduction to Metamathematics* 一书中（1952 年 Van Nostrand 公司出版）。但其实这个概念早在 20 世纪 40 年代初就已形成。到了 70 年代，随着 Unix 操作系统及其实用程序 sed、grep 等问世，正则表达式得到了计算机科学家更为广泛的使用。Unix 操作系统是美国电话电报公司下属贝尔实验室的 Brian Kernighan、Dennis Ritchie、Ken Thompson 以及其他工作人员的杰作。

据我所知，最早出现正则表达式的计算机应用程序是 QED 编辑器。QED 是 Quick Editor 的缩写，它是为运行在 Scientific Data Systems 公司[1] SDS 940 计算机中的 Berkeley Timesharing System 编写的。1970 年的记录显示，QED 是由 Ken Thompson 在之前 MIT 的 Compatible Time-Sharing System 中另外一个编辑器基础上重写而成的。从此，计算技术领域有了真正的正则表达式实现。（附录中的表 A-1 列出了 QED 的正则表达式特性。）

注 1：Scientific Data Systems（英文缩写 SDS），是 Max Palevsky 于 1961 年在美国成立的一家计算机公司，也是最早在计算机设计中使用集成电路和硅晶体管的公司。SDS 计算机主要针对大型科学计算，物美价廉。“太空竞赛”期间 NASA 曾购买了很多台 SDS 计算机。SDS 在 1969 年被施乐（Xerox）公司收购，1975 年由于管理不善和销售下滑被关闭。在施乐管理期间，该公司一度被称为 XDS。——编者注

本书中用来展示示例的工具很多，但多数都容易获取，而且也很实用。只有少数工具目前还没有好用的 Windows 版本。如果你觉得哪个工具不好用，完全可以不用。但要真正学习正则表达式，我还是建议在 Unix 环境中学习。我使用 Unix 环境长达 25 年，每天仍然能够学到不少新东西。

> “不懂Unix的人注定还要重新发明一个蹩脚的Unix。”
>
> ——Henry Spencer [2]

部分工具可以通过浏览器在线使用，这对于许多读者是十分方便的。其余的工具需要使用命令行和 shell 脚本，还有一些工具是在桌面上运行的。如果你手头没有这些工具，从网上下载也很方便。其中大多数工具是免费的，偶尔有需要付费的也不贵。

本书中不会出现很多专业术语。我会在必要的时候告诉你正确的术语，但这种情况很少。因为多年的经验表明，专业术语常会造成障碍。换句话说，我会尽可能用通俗易懂的语言描述正则表达式，以免你晕头转向不知所措。因为本书的理念是“略知大概，即可实践”。

正则表达式的实现多种多样。你会发现在 vi（vim）、grep 及 sed 等 Unix 命令行工具中使用的正则表达式也可以在其他程序中见到。各种程序设计语言都支持正则表达式，比如 Perl（当然啦 [3]）、Java、JavaScript、C#、Ruby 等。就连 XSLT 2.0 这样的声明式语言中也有正则表达式。你还会发现 Notepad++、Oxygen 及 TextMate 等应用程序同样支持正则表达式。

大多数正则表达式实现各有异同。本书不会逐一讨论它们的差异，但也会涉及一些。如果要我把所有实现的全部不同点都列出来，恐怕非得把我累吐了血不行。所以我在本书中就不纠缠这些细节了。总而言之，如果你期待一本正则表达式的入门书，那就选这本吧。

注 2：Henry Spencer 是加拿大程序员，著名正则表达式库 regex 的作者。这个正则表达式库被许多程序包或编程语言采用，比如 Perl、Tcl 和 MySQL，等等。在多伦多大学工作期间，Henry Spencer 从 1981 年开始运作美国之外的第一个 Usernet 站点。这个站点后来被谷歌收购，作为 20 世纪 80 年代 Usernet 的公开档案。另外，他还写过“10 Commandments for C Programmers”（C 程序员十诫，http://www.seebs.net/c/10com.html）。——编者注

注 3：Perl，后来被人们解释为 Practical Extraction and Reporting Language 的缩写。由这个非官方的“全称”——实用提取和报告语言——可知，Perl 在处理文本文件和生成报表方面是非常强大的。1987 年，Larry Wall 在美国宾夕法尼亚州蓝铃（Blue Bell）地区的 Unisys 公司当程序员的时候发明了 Perl。在该语言后来的发展中，正则表达式功能得到不断丰富和加强，最终成为 Perl 独树一帜的招牌特色。
——编者注

目标读者

本书适合从零开始学习正则表达式的读者。如果你准备开始学习正则表达式或编写程序，本书就是很好的起点。换句话说，所有听说过正则表达式且对其非常感兴趣，但还没有真正理解正则表达式的人，都是这本书的目标读者。如果你属于这种情况，本书很适合你。

我将采取由简到繁的顺序来介绍正则表达式的特性，也就是先从简单的开始。

如果你已经对正则表达式及其用法有所了解，或者已是位编程老手，本书可能不适合你。本书面向需要指导的初学者。如果你写过一些正则表达式，但希望加深对基本概念的理解，也可以阅读本书，只是我们的节奏可能比你想像的要慢一些。

我在此推荐几本学习完本书后应该阅读的书。第一本是 Jeff Friedl 的 *Mastering Regular Expressions, Third Edition*（参见 http://shop.oreilly.com/product/9781565922570.do）。这本书对正则表达式进行了全面阐述，强烈推荐读者阅读。Jan Goyvaerts 和 Steven Levithan 执笔的 *Regular Expressions Cookbook*（参见 http://shop.oreilly.com/product/9780596520694.do）也是一本相当不错的书[4]。Jan Goyvaerts 开发的 RegexBuddy 是一个强大的桌面应用（参见 http://www.regexbuddy.com），而 Steven Levithan 开发了我们第 1 章就会用到的在线正则表达式处理程序 RegexPal（参见 http://www.regexpal.com）。

阅读要求

为了更好地学习本书，你需要使用 Unix 或 Linux 操作系统上的一些工具。这些工具在 Mac 上的 Darwin 系统（BSD 在 Mac 上的衍生系统）、Windows 电脑中运行的 Cygwin（参见 http://www.cygwin.com 及 http://www.gnu.org）中都能找得到。

本书提供了大量示例供你实验，只是看一遍印象不会深刻。要真正掌握正则表达式，就应该按照这些示例的步骤自己操作，最好把所有示例都过一遍。最好的学习方式是亲身实践，而不是做个旁观者。你得通过本书学会使用高亮显示匹配结果以验证正则表达式的网站、强大的 Unix 命令行工具、分析正则表达式或用正则表达式搜索文本的桌面程序。

Github 上托管着本书的示例代码（https://github.com/michaeljamesfitzgerald/Introducing-Regular-Expressions）。另外，通过 http://examples.oreilly.com/0636920012337/examples.zip

注 4：这两本书的中文版分别是《精通正则表达式（第 3 版）》（电子工业出版社）和《正则表达式经典实例》（人民邮电出版社）。——编者注

也可以下载到本书所有的示例和测试文件[5]。学习本书之前，你最好先在自己的计算机上创建一个新文件夹，并将这些文件保存到该文件夹中。

致谢

首先，请允许我再次向本书的编辑、O’Reilly 出版社的 Simon St. Laurent 表达深深的谢意。Simon 耐心认真，没有他就不会有这么出色的书。同时感谢 Seara Patterson Coburn 和 Roger Zauner，他们的审读意见提升了本书的质量。另外，一如既往感谢我的挚爱 Cristi，你就是我 *raison d’être*（《存在的理由》）。

Safari® Books Online



Safari Books Online（www.safaribooksonline.com）是应需而变的数字图书馆。它同时以图书和视频的形式出版世界顶级技术和商务作家的专业作品。

Safari Books Online 是技术专家、软件开发人员、Web 设计师、商务人士和创意人士开展调研、解决问题、学习和认证培训的第一手资料。

对于组织团体、政府机构和个人，Safari Books Online 提供各种产品组合和灵活的定价策略。用户可通过一个功能完备的数据库检索系统访问 O’Reilly Media、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit Press、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kaufmann、IBM Redbooks、Packt、Adobe Press、FT Press、Apress、Manning、New Riders、McGraw-Hill、Jones & Bartlett、Course Technology 以及其他几十家出版社的上千种图书、培训视频和正式出版之前的书稿。要了解 Safari Books Online 的更多信息，我们网上见。

排版约定

本书使用的排版约定如下。

- 楷体
  表示新的术语。

注 5：读者也可以从图灵社区本书网页（http://www.ituring.com.cn/book/955）随书下载部分下载本书示例代码。——编者注

- 等宽字体

  表示程序片段，也用于正文中表示程序中使用的变量、函数名、命令行代码、环境变量、语句和关键词等代码文本。



这个图标代表小窍门、建议或说明。

联系我们

请把对本书的评价和问题发给出版社。

美国：

O’Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）
奥莱利技术咨询（北京）有限公司

O’Reilly 的每一本书都有专属网页，你可以在那儿找到本书的相关信息，包括勘误表、示例代码以及其他信息。本书的网站地址是：

http://oreil.ly/intro_regex

中文版地址：

http://www.oreilly.com.cn/index.php?func=book&isbn=978-7-115-31149-8

对于本书的评论和技术性问题，请发送电子邮件到：

bookquestions@oreilly.com

要了解更多 O’Reilly 图书、培训课程、会议和新闻的信息，请访问以下网站：

http://www.oreilly.com

我们在 Facebook 的地址如下：

http://facebook.com/oreilly

请关注我们的 Twitter 动态：

http://twitter.com/oreillymedia

我们的 YouTube 视频地址如下：

http://www.youtube.com/oreillymedia

示例代码

本书就是要帮读者解决实际问题的。也许你需要在自己的程序或文档中用到本书中的代码。除非大段大段地使用，否则不必与我们联系取得授权。因此，用本书中的几段代码写成一个程序不用向我们申请许可。但是销售或者分发 O'Reilly 图书随附的代码光盘则必须事先获得授权。引用书中的代码来回答问题也无需我们授权。将大段的示例代码整合到你自己的产品文档中则必须经过许可。

使用我们的代码时，希望你能标明它的出处。出处一般要包含书名、作者、出版商和 ISBN，例如"*Introducing Regular Expressions* by Michael Fitzgerald (O'Reilly). Copyright 2012 Michael Fitzgerald, 978-1-4493-9268-0"。

如果还有其他使用代码的情形需要与我们沟通，可以随时与我们联系：permissions@oreilly.com。

第 1 章

什么是正则表达式

正则表达式是一种特殊的字符串模式，用于匹配一组字符串。它最早出现于 20 世纪 40 年代，当时用来描述正则语言，而到 20 世纪 70 年代才真正出现在程序设计领域。据我所知，正则表达式首次出现在 Ken Thompson 编写的 QED 文本编辑器中。

> “正则表达式是描述一组字符串特征的模式，用来匹配特定的字符串。”
>
> ——Ken Thompson

正则表达式后来成为 ed、sed 和 vi（vim）编辑器、grep、AWK 等 Unix 操作系统衍生出的工具集中重要的组成部分。但这些工具实现正则表达式的方式并不完全一致。



本书使用归纳方式讲述，也就是说，会从特例讲起，最终归结到一般情况。所以我们不会先陈述观点，然后举例，而是先为大家展示示例，然后归纳出一般性结论。本书的风格是“实践出真知”。

正则表达式素以“坑多”而闻名，但是这跟你的学习方法有关。大家通常会从下面这个简单的正则表达式开始学习：

```
\d
```

——这是匹配 0 到 9 范围内的任意数字的字符组简写式——然后过渡到更为复杂的内容，比如：

```
^((\d{3})|^\d{3}[.-]?)?\d{3}[.-]?\d{4}$
```

这是本章最后才会讲到的一个还算比较健壮的正则表达式，它可以匹配 10 位的北美电话号码。无论区号是否加括号（如果加，则左、右括号必须成对出现），数字间是否有点号（句点）或连字符，它都可以匹配。



第 10 章会展示一个更复杂的匹配电话号码的正则表达式，但本章你只要学会上面这个就行了。

现在还不理解也没关系，本章会将整个表达式的内容分拆开一点一点地教给你。只要你跟着这里的示例做（当然包括本书其他示例），编写正则表达式很快就可以习惯成自然。你准备好了吗？

我有时会将本书中的 Unicode 字符用它们的代码点（4 位的十六进制数）来表示。这些代码点以 *U+0000* 的形式出现。比如，U+002E 表示点号的代码点。

1.1 从Regexpal开始

首先介绍一下 Regexpal 网站（ http://www.regexpal.com）。在 Google Chrome 或者 Mozilla Firefox 之类的浏览器中打开该网站，可以看到如图 1-1 所示的网站页面。

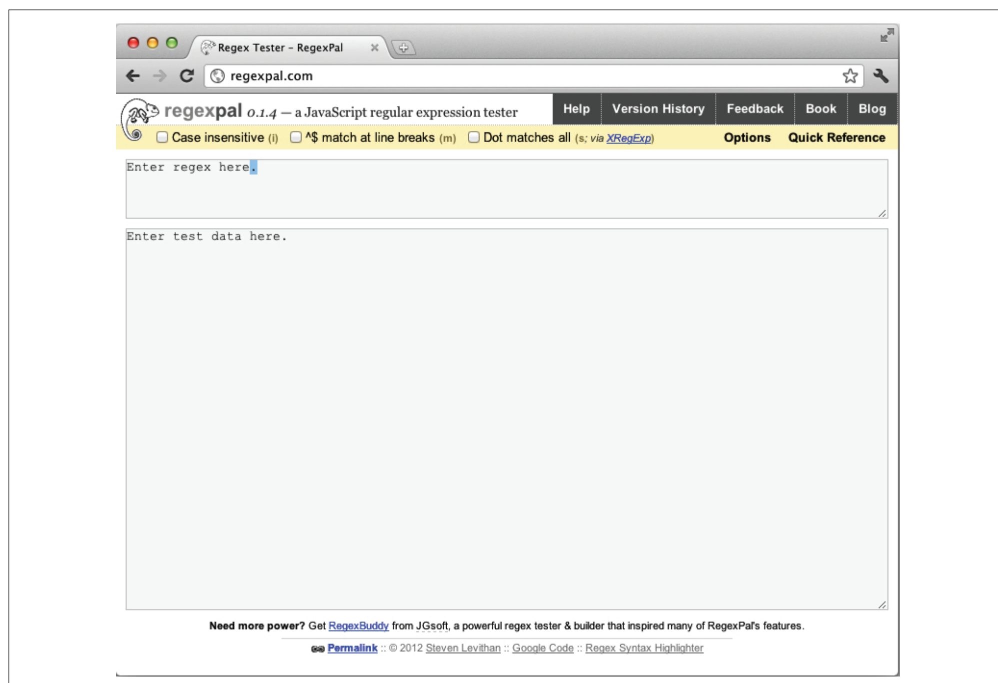


图 1-1：在谷歌 Chrome 浏览器中打开 Regexpal 网站

接近浏览器窗口的顶部有一个文本区，下方还有一个更大的文本区。顶部的文本区是用来输入正则表达式的，下方的文本区是用来输入主题或目标文本的。目标文本即要从中匹配字符串的文本。



本书正文每一章的结尾都有一节“技术备忘录”。这些备忘录提供与所在章节讨论的技术有关的额外信息，并告诉你到哪里可以找到更多的相关信息。这种方法既不会打断正文的节奏，又能为读者多提供一些扩展信息。

1.2 匹配北美电话号码

现在我们要写一个正则表达式匹配北美电话号码。在 Regexpal 的下方文本框中键入以下电话号码：

```
707-827-7019
```

知道这是哪里的电话号码吗？这是 O’Reilly Media 的电话号码。

接下来用正则表达式匹配这个号码。有很多方法都可以做到，但首先只要在上方的文本框键入这个号码本身，也就是与下方文本框的内容完全一致即可（先忍耐一下，别泄气）：

```
707-827-7019
```

这时，你应该看到下方文本框中的电话号码从头到尾都以黄色被高亮显示。如果你的结果也是这样（如图 1-2 所示），那么一切正常。



本书中提及的有关图片或者屏幕截图中的颜色，比如 Regexpal 中的黄色高亮效果，应该只能在线或者从本书的电子版中看到，而不会出现在纸质版本中。如果你在阅读本书的纸质版，我所提及的颜色在你看来将会是灰阶的，对此我非常抱歉。

刚刚你所写的正则表达式是用字符串字面值（string literal）来匹配目标字符串的。所谓字符串字面值，就是字面上看起来是什么就是什么。

现在将上方文本框的号码删除，然后只键入数字 7。你看到什么了？现在只有数字 7 高亮显示。正则表达式中的字面值（数字）7 与目标文本中数字 7 的四个实例匹配。

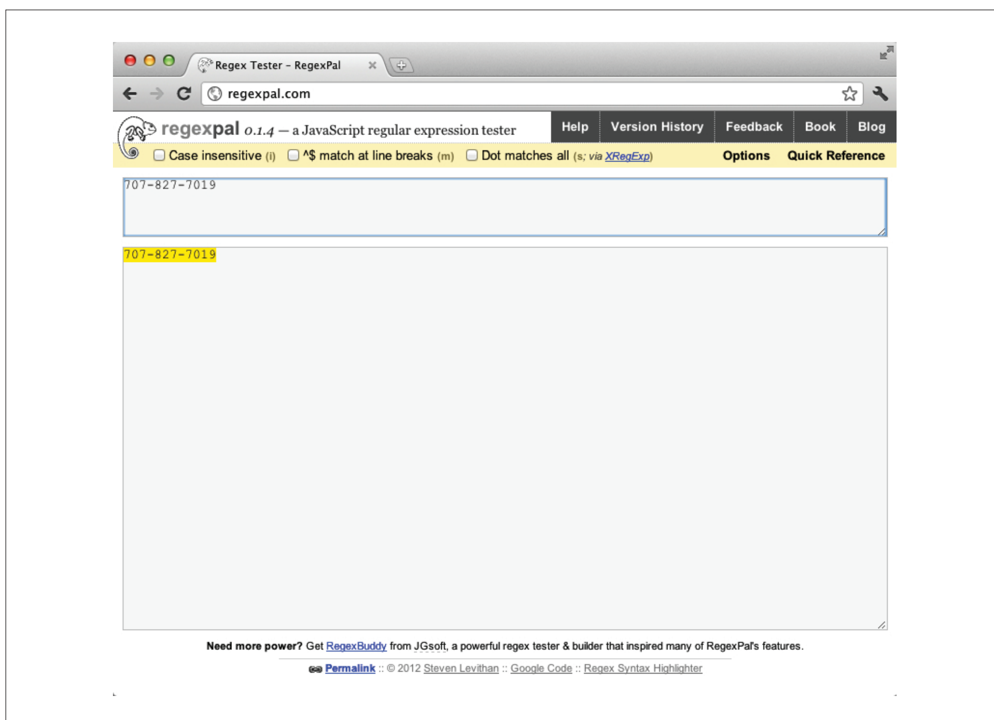


图 1-2：Regexpal 中高亮显示的十位电话号码

1.3　用字符组来匹配数字

如果想同时匹配电话号码中的所有数值或者只匹配特定的数值，该怎么办呢？

在上方的文本框中尝试以下表达式：

```
[0-9]
```

下方文本框中所有的数值（准确地说，应该是数字）以黄色和蓝色交替高亮显示。正则表达式 `[0-9]` 对正则表达式处理器传递的信息是“匹配 0 到 9 范围内的任意数字”。

正则表达式将方括号视为特殊的*元字符*（metacharacter），因此方括号不参与匹配。元字符是在正则表达式中有特殊含义的字符，也是保留字符。`[0-9]` 这种形式的正则表达式称做*字符组*（character class），有时也叫*字符集*（character set）。

可以对数字的范围进行进一步限定。用更具体的一组数字也能得到同样的结果，比如：

```
[012789]
```

这个字符组只会匹配列出的数字，即 0、1、2、7、8、9。在上方的文本框中试一试，下方文本框中的相应数字同样会被交替标亮。

要匹配任意 10 位以连字符分隔的北美电话号码，可以使用以下正则表达式：

```
[0-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]
```

这是可以的，但太长了。更好的方法是采用简写形式。

1.4　使用字符组简写式

就像在本章开始看到的那样，`\d` 可以像 `[0-9]` 一样匹配任意阿拉伯数字。请在上方文本框内试一试，和之前的表达式一样，下方的数字都被标亮了。这种正则表达式叫做*字符组简写式*（character shorthand），也叫*转义字符*（character escape）。但后一种称谓很容易造成误解，我会尽量不用。至于原因，稍后我再解释。

可以使用以下表达式来匹配电话号码中的任意数字：

```
\d\d\d-\d\d\d-\d\d\d\d
```

重复 `\d` 三次和四次就可以分别匹配三个和四个数字。该表达式中的连字符是一个字面值，因此会被原样匹配。

除了和上面表达式一样，使用连字符本身（-）来匹配连字符之外，也可以用转义的大写 D（`\D`），它匹配任何一个*非数字*字符。

以下示例使用了 `\D`（而没有使用连字符本身）来匹配连字符：

```
\d\d\d\D\d\d\d\D\d\d\d\d
```

这次整个电话号码包括连字符又都被标亮了。

1.5　匹配任意字符

还可以用点号（.）来匹配那些讨厌的连字符：

```
\d\d\d.\d\d\d.\d\d\d\d
```

点号（英文句号）是一个通配符，可以匹配任意字符（但某些情况下不能匹配行起始符）。以上示例中的正则表达式匹配了连字符，但它也可以匹配百分号（`%`）：

```
707%827%7019
```

或者是竖线（|）：

```
707|827|7019
```

亦或其他字符。



如前所述，点号一般不匹配行起始符，比如换行符（U+000A）。然而，有很多方法可以使点号匹配行起始符，之后我会展示。这通常叫做 dotall 选项。

1.6　捕获分组和后向引用

本节我们使用捕获分组（capturing group）来匹配电话号码中的某一部分。然后使用后向引用（backreference）对分组中的内容进行引用。要创建捕获分组，先将一个 `\d` 放在一对圆括号中，这样就将它放入了一个分组中，后面可以用 `\1` 来对捕获的内容进行后向引用：

```
(\d)\d\1
```

`\1` 对括号内分组捕获的内容进行了反向引用。这个正则表达式匹配的是区号 707。以下是对该表达式的详细分析：

- `(\d)` 匹配第一个数字并将其捕获（数字 7）；
- `\d` 匹配第二个数字（数字 0）但没有捕获，因为没有括号；
- `\1` 对捕获的数字进行反向引用（数字 7）。

这个正则表达式只匹配了区号。如果你还没有完全理解，请不要担心。本书后面会介绍很多有关捕获分组的示例。

现在可以用一个分组和几个后向引用对整个电话号码进行匹配：

```
(\d)0\1\D\d\d\1\D\1\d\d\d
```

但这还不够简洁美观。下一节我们会尝试更好的方法。

1.7　使用量词

现在用另一种语法来匹配电话号码：

```
\d{3}-?\d{3}-?\d{4}
```

花括号中的数字表示待查找的数字出现的次数。包含数字的花括号是一种量词

(quantifier)。花括号本身用做元字符。

问号是另一种量词，在以上表达式中表示连字符是可选的。也就是说，连字符可以不出现或只出现一次。还有其他的量词，例如加号（+）表示“一个或多个”，星号（*）表示“零个或多个”。

使用量词能让正则表达式变得更简洁：

```
(\d{3,4}[.-]?)+
```

对，加号表示出现一次或多次。这个正则表达式表示括号里的模式出现一次或多次，括号里的模式匹配三位或四位数字，后跟一个连字符或一个点号。

你有没有头晕呢？我希望没有。下面逐一解释表达式中的每一项：

- 左圆括号 (为捕获分组的起始符；
- 反斜杠 \ 为字符组简写式的起始符（对之后的字符进行转义）；
- 字符 d 为字符组简写式的结束符（d 匹配 0 到 9 范围内的任意数字）；
- 左花括号 { 为量词起始符；
- 数字 3 为匹配的最小数量；
- 逗号 , 隔开不同的数量；
- 数字 4 为匹配的最大数量；
- 右花括号 } 为量词的结束符；
- 左方括号 [为字符组的起始符；
- 点号 .（匹配点号本身）；
- 连字符 - 匹配连字符的本身；
- 右方括号] 为字符组的结束符；
- 问号 ? 表示量词“零个或一个”；
- 右圆括号) 为捕获分组的结束符；
- 加号 + 表示量词“一个或多个”。

这个表达式能用但不完全对，因为它只能匹配 3 位或 4 位的数字，而不管是否符合电话号码的格式。好吧，犯错会让我们印象深刻，进步得更快。

我们来改进一下：

```
(\d{3}[.-]?){2}\d{4}
```

这个表达式匹配的字符串是连续两个无括号的三位数字，每三位数字后可以带连字符也可以不带，最后是一个四位数字。

1.8 括选文字符

最后这个正则表达式表示第一个 3 位数字可以带也可以不带括号，即区号是可选的：

```
^(\(\d{3}\)|^\d{3}[.-]?)?\d{3}[.-]?\d{4}$
```

为了便于理解，我们再依次看一下表达式中的各项：

- 出现在正则表达式起始位置或者竖线（|）之后的脱字符 ^，表示电话号码会出现在一行的起始位置；
- 左括号 (为捕获分组的起始符；
- \(表示左括号本身；
- \d 匹配一位数字；
- \d 之后的 {3} 是量词，表示匹配三位数字；
- \) 匹配右括号本身；
- 竖线符 | 表示选择，也就是从多个可选项中选择一个，换句话说，它表示“匹配一个不带括号的区号或一个带括号的区号”；
- 脱字符 ^ 匹配行起始位置；
- \d 匹配一位数字；
- {3} 是表示匹配三位数字的量词；
- [.-]? 匹配一个可选的点号或连字符；
- 右括号) 为捕获分组的结束符；
- 问号? 表示分组可选，即分组中的前缀可有可无；
- \d 匹配一位数字；
- {3} 表示匹配三位数字的量词；
- [.-]? 匹配另一个可选的点号或连字符；
- \d 匹配一位数字；
- {4} 是表示匹配四位数字的量词；
- 美元符 $ 匹配行结束位置。

这个表达式最终匹配十位的北美电话号码，而且括号、连字符或者点号都是可选的。你可以试试不同格式的电话号码，看看它能否匹配。



以上正则表达式中的捕获分组并不是必需的。分组是必要的，但是捕获不需要。更好的方法是使用非捕获分组。在本书最后一章中我们再次讨论这个正则表达式时，你自然就理解了。

1.9 应用举例

本章最后，我们在几个应用程序里测试一下匹配电话号码的正则表达式。

TextMate 是一个只在 Mac 上运行的文本编辑器，它采用与 Ruby 语言相同的正则表达式程序库。你可以通过 Find（查找）对话框使用正则表达式，如图 1-3 示。将 Regular expression 旁的复选框选中。

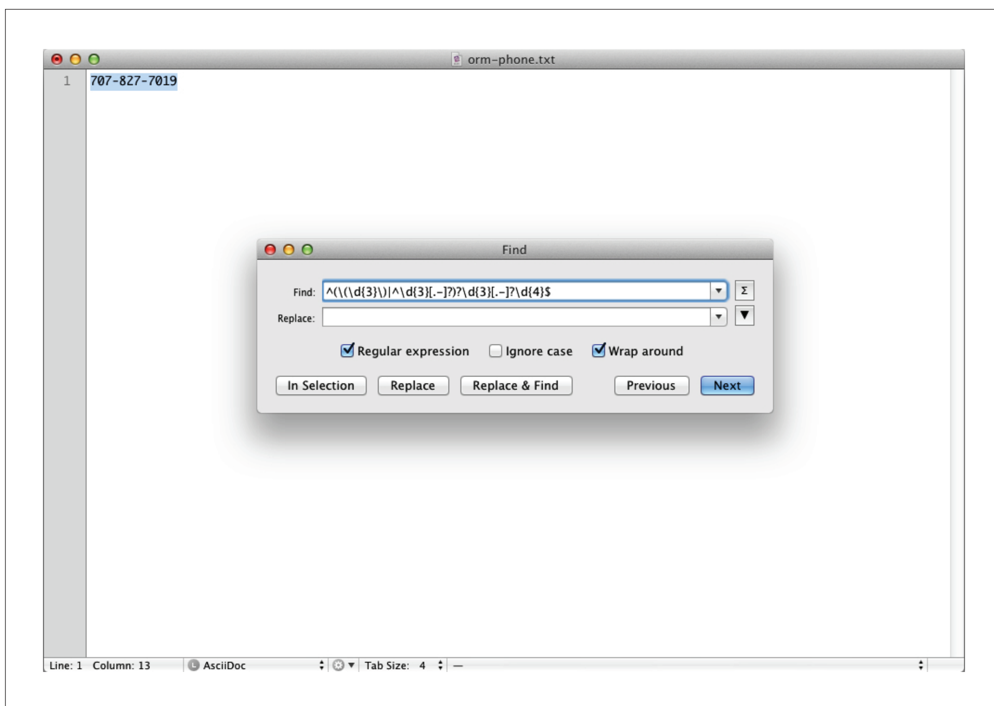


图 1-3：在 TextMate 中测试正则表达式

Notepad++ 是运行于 Windows 上的一个常用的免费文本编辑器，它采用了 PCRE（Perl Compatible Regular Expression，Perl 兼容正则表达式）库。在勾选了 Regular expression 旁的单选按钮之后，就可以用正则表达式进行查找和替换了（参见图 1-4）。

Oxygen 是个流行且强大的 XML 编辑器，它使用 Perl 5 的正则表达式语法。可以通过图 1-5 中的查找和替换对话框或者通过 XML Schema 的正则表达式构建工具使用正则表达式。要在查找和替换对话框里使用正则表达式，则要勾选 Regular expression 旁的复选框。

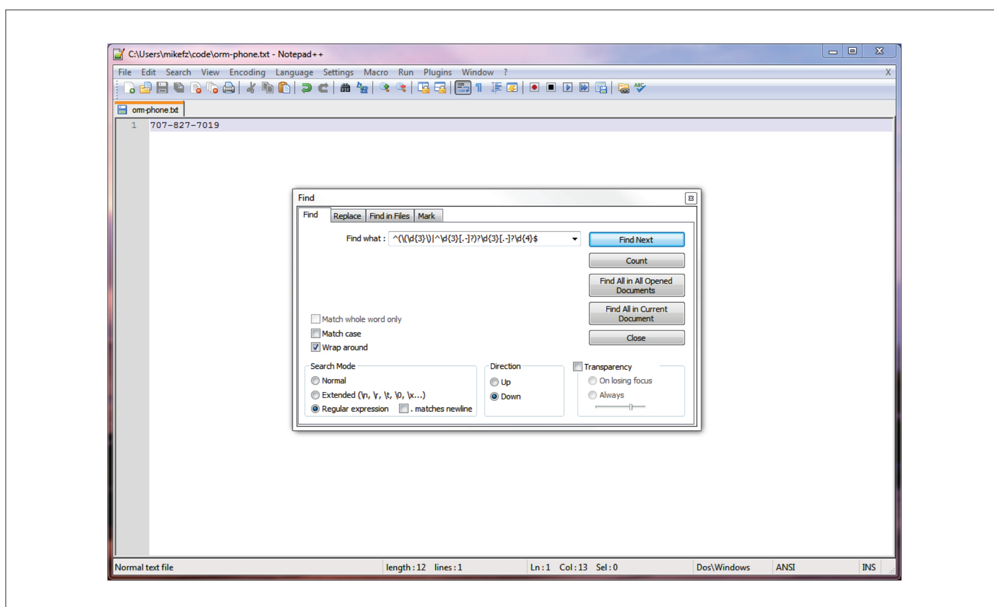


图 1-4：在 Notepad++ 中测试正则表达式

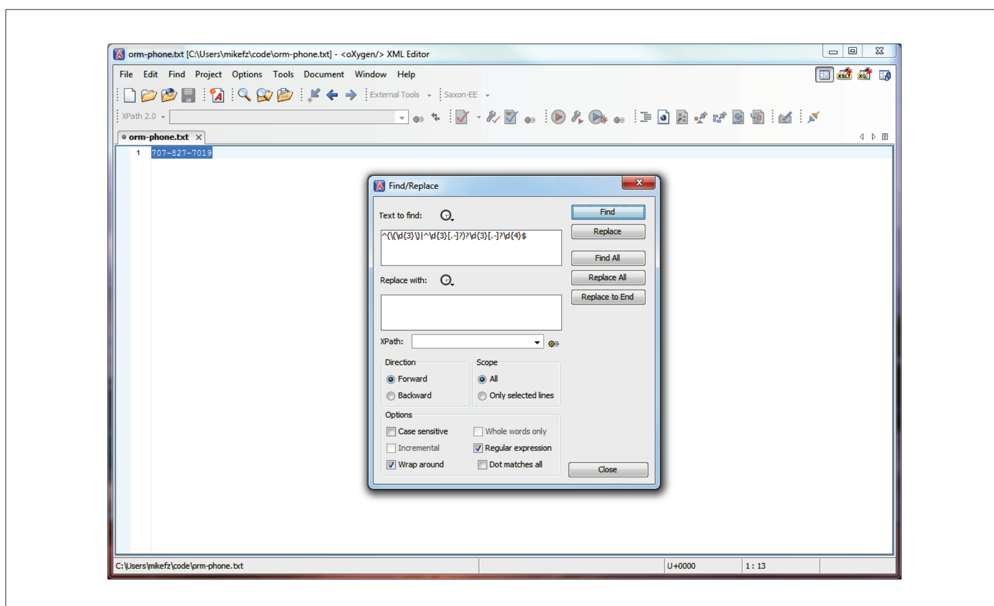


图 1-5：在 Oxygen 中测试正则表达式

对正则表达式的简介到此为止。恭喜你，你在本章已经接触了不少基础内容。下一章的重点是简单的模式匹配。

1.10 本章所学

- 什么是正则表达式
- 如何使用简单的正则表达式处理器 Regexpal
- 如何匹配字符串字面值
- 如何使用字符组匹配数字
- 如何使用字符组简写式匹配一位数字
- 如何使用字符组简写式匹配一个非数字字符
- 如何使用捕获分组和后向引用
- 如何匹配确切数量的字符串
- 如何匹配出现零次或一次的字符（可选字符）和出现一次或多次的字符
- 如何匹配行起始位置或行结束位置的字符串

1.11 相关资源

- Regexpal（http://www.regexpal.com）是一个在线的用 JavaScript 实现的正则表达式处理器。它并不是正则表达式的完整实现，因此功能并不完整；但是它是个简洁易用的学习工具，很容易上手。
- 可以从 https://www.google.com/chrome 下载 Chrome 浏览器，或者从 http://www.mozilla.org/en-US/firefox/new/ 下载 Firefox 浏览器。
- 为什么有那么多编写正则表达式的方法？一个原因是正则表达式具有*可组合性*（composability）。对于一种具备*可组合性*的语言（James Clark 很好地解释了这种特性，参见 http://www.thaiopensource.com/relaxng/design.html#section5），不管是形式语言、程序设计语言还是模式语言，都可以很容易地将其原子部分和构造方法用各种不同的方式重新组合。只要掌握了正则表达式的所有“原子部分”，你就会发现没什么字符串是匹配不出来的。
- TextMate 可以从 http://www.macromates.com 获取。有关 TextMate 中正则表达式的更多信息，请参考 http://manual.macromates.com/en/regular_expressions。
- 有关 Notepad++ 的更多信息，请参见 http://notepad-plus-plus.org。有关如何在 Notepad 中使用正则表达式，请参考 http://sourceforge.net/apps/mediawiki/notepad-plus/index. php?title=Regular_Expressions。
- 访问 http://www.oxygenxml.com 可以看到更多有关 Oxygen 的内容。有关在查找和替换对话框中使用正则表达式的信息，请参见 http://www.oxygenxml.com/doc/ug-editor/topics/find-replace-dialog.html。有关 XML Schema 的正则表达式构建工具的信息，请参见 http://www.oxygenxml.com/doc/ug-editor/topics/XML-schema-regexp-builder.html。

第2章
简单的模式匹配

正则表达式唯一的用途就是在文本中匹配和寻找模式，模式可以简单也可以复杂。本章将会介绍一些采用以下概念匹配模式的简单方法：

- 字符串字面值；
- 数字；
- 字母；
- 任意字符 。

第 1 章我们使用了 Steven Levithan 的 RegexPal 演示了正则表达式的作用。本章我们将使用 Grant Skinner 的 RegExr 网站，网址为 http://gskinner.com/RegExr（如图 2-1 所示）。



本书对正则表达式的讲解会逐步深入。在这个过程中，也希望你能逐步深入地理解正则表达式语法。我想说的是，当你遇到新事物时要主动尝试。尝试、失败、掌握、进步。通过实践来学习是不会有错的。

在深入学习之前，我们先看看 RegExr 为我们提供了什么好东西。在 RegExr 界面右上方有三个标签，请注意 Samples 和 Community。Samples 标签提供正则表达式语法方面的许多帮助信息，Community 标签则展示了大量由别人提供且得到评分的正则表达式。这些标签中的很多信息可能对你非常有用。把鼠标光标放在 RegExr 的正则表达式或者目标文本上，就会弹出一些有用的信息。这些都是我非常喜欢 RegExr 这个在线正则表达式验证工具的原因。

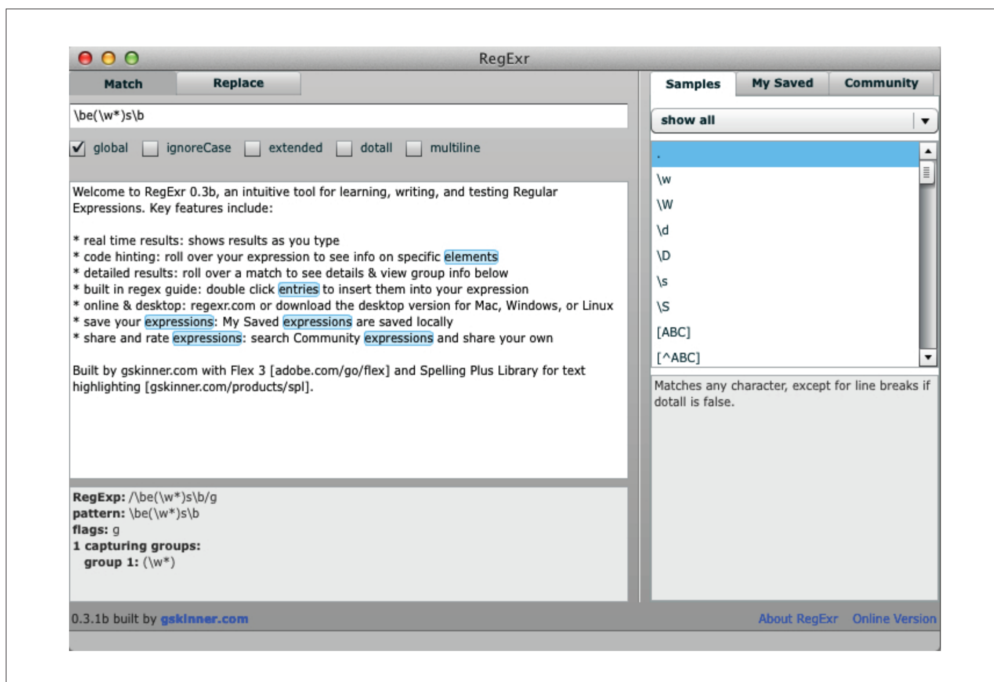


图 2-1：在 FireFox 中打开 Grant Skinner 的 RegExr

本章以《抒情歌谣集》（*Lyrical Ballads*，London, J. & A. Arch, 1798）中收录的塞缪尔・泰勒・柯勒律治的一首诗“The Rime of the Ancient Mariner”为主要示例文本。接下来的几章中都会用到这首诗，开始是普通文本版，之后过渡到由 HTML5 标记的版本。该诗的全文保存在名为 rime.txt 的文件中；本章使用的 rime-intro.txt 文件只包括该诗的前几行。

以下几行是 rime-intro.txt 的内容：

```
THE RIME OF THE ANCYENT MARINERE, IN SEVEN PARTS.

ARGUMENT.

How a Ship having passed the Line was driven by Storms to the cold Country
towards the South Pole; and how from thence she made her course to the tropical
Latitude of the Great Pacific Ocean; and of the strange things that befell;
and in what manner the Ancyent Marinere came back to his own Country.

I.

1       It is an ancyent Marinere,
2         And he stoppeth one of three:
3       "By thy long grey beard and thy glittering eye
4         "Now wherefore stoppest me?
```

将这些内容复制并粘贴到 RegExr 中下方的文本框中。可以在 Github 中找到 rime-intro.txt 文件，地址是 https://github.com/michaeljamesfitzgerald/Introducing-Regular-Expressions；也可以在通过 http://examples.oreilly.com/9781449392680/examples.zip 下载到的压缩包中找到该文件。还可以在 Project Gutenberg 中找到这首诗在线文本，只是没有行号（参见 http://www.gutenberg.org/ebooks/9622）。

2.1 匹配字符串字面值

正则表达式最为直接和明显的功能就是用一个或多个字符字面值来匹配字符串。

匹配字符串字面值的方法就是使用普通的字符 。这听起来是否有些熟悉呢？这就和你在 Word 等字处理程序中使用查找或者在搜索引擎中输入关键字类似。当你以逐个字符对应的方式查找文本字符串的时候，就是在用字符串字面值查找。

举个例子，如果你要匹配以上诗文中开头部分的单词 Ship，只需在 RegExr 上方的文本框中键入 Ship，该单词就会在下方的文本框中标亮（首字母要大写）。

下方文本框中是否有淡蓝色标亮部分？应当能看到，如果你没有看到，请检查一下输入的字面值。



默认情况下，RegExr 中的字符串匹配是区分大小写的。若要不区分大小写，则要勾选 RegExr 左上方的 ignoreCase（忽略大小写）旁的复选框。选中该复选框后，目标文本中的 Ship 或 ship 都会匹配。

2.2 匹配数字

在 RegExr 中左上方的文本框中，输入以下字符组简写式来匹配数字：

```
\d
```

因为默认勾选了 global（全局匹配）复选框，这将会匹配下方文本区域中所有的阿拉伯数字。取消选择该复选框后，`\d` 只会匹配第一个出现的数字（参见图 2-2）。

现在用字符组替代 `\d` 来匹配相同的内容。在 RegExr 的上方文本框中输入以下范围的数字：

```
[0-9]
```

如同在图 2-3 中看到的，虽然语法不一样，但 `\d` 和 `[0-9]` 的效果是一样的。

图 2-2：在 RegExr 中用 `\d` 匹配所有的数字

图 2-3：在 RegExr 中用 `[0-9]` 匹配所有的数字



第 5 章将会学习更多有关字符组的内容。

字符组 [0-9] 表示范围，这意味着它会匹配 0 至 9 范围内的数字。你也可以列出 0 至 9 范围内的所有数字来进行匹配：

```
[0123456789]
```

如果只想匹配 0 和 1 两个数字，可以使用这个字符组：

```
[01]
```

请在 RegExr 中尝试一下 [12] 并看看结果。使用字符组可精确匹配字符。数字的字符组简写式 \d 更为简短，但却没有字符组强大、灵活。在无法使用 \d 时（不是所有情况下都支持这种方式），或者想匹配特定数字时，就需要使用字符组；合适的时候可以使用 \d，因为它更简短。

2.3 匹配非数字字符

通常可以将简写式取反，取反的结果就是排除。比如，要匹配非数字字符，可使用包含以下大写字母 D 的简写式：

```
\D
```

请在 RegExr 中试一试。大写字母 D 取代小写字母 d，就会匹配非数字字符（如图 2-4 所示）。该简写式与以下字符组取反的作用相同（字符组取反的意思其实就是“不匹配这些”或“匹配除这些以外的内容”）：

```
[^0-9]
```

下面这个表达式作用也一样：

```
[^\d]
```

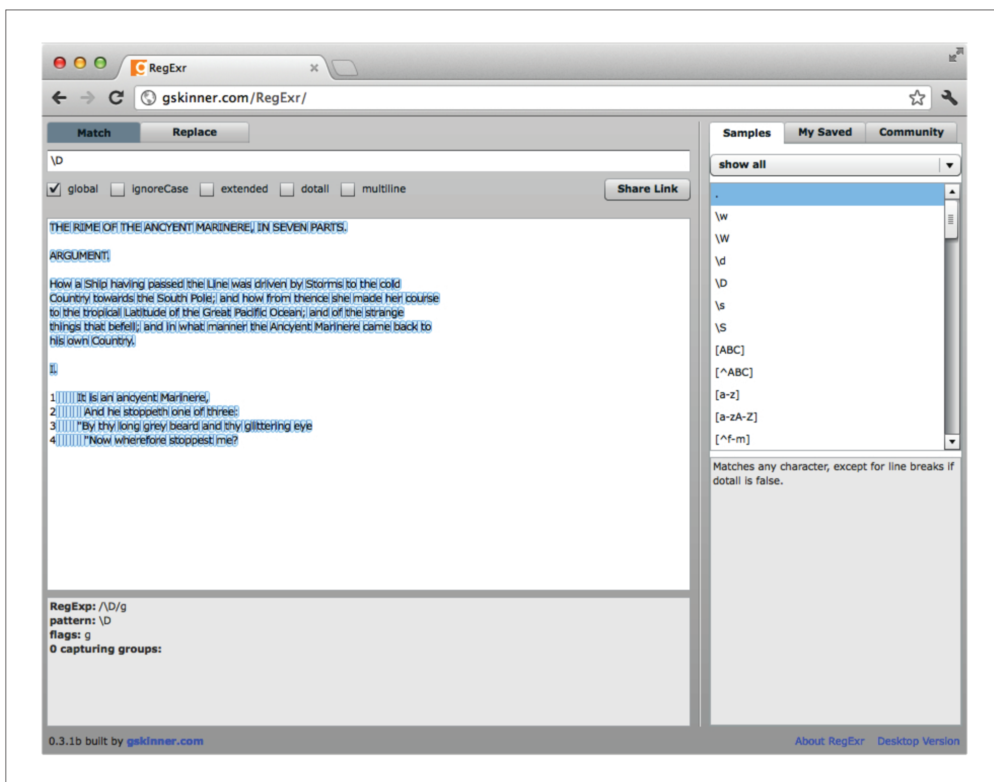


图 2-4：在 RegExr 中用 `\D` 匹配非数字字符

2.4　匹配单词和非单词字符

在 RegExr 中，将 `\D` 替换为：

```
\w
```

这个简写式将匹配所有的单词字符（前提是勾选 global 选项）。`\D` 与 `\w` 的区别是 `\D` 会匹配空格、标点符号（引号、连字符、反斜杠、方括号）等字符，而 `\w` 却不会，它只匹配字母、数字和下划线。

在英语环境中，与 `\w` 匹配相同内容的字符组为：

```
[_a-zA-Z0-9]
```



我们会在第 6 章学习怎样匹配非英文字符。

现在用大写字母 W 匹配非单词字符：

```
\W
```

在本示例中，这个简写式匹配空格、标点以及其他非字母、非数字字符。使用以下字符组也可以匹配相同的内容：

```
[^_a-zA-Z0-9]
```

字符组允许你匹配更多类型的字符，但有时你不想而且也没有必要键入所有字符。这也就是“按键次数最少则胜”的原则。但有时你确实需要将所有的字符键入才能得到准确的结果。反正你自己决定。

轻松一下，在 RegExr 中试一下：

```
[^\w]
```

以及

```
[^\W]
```

你看到匹配结果的差异了吗？

表 2-1 提供了更多的字符简写式。不过并不是所有的正则表达式处理器都能识别这些简写式。

表2-1：字符简写式

字符简写式	描　　述	字符简写式	描　　述
\a	报警符	\w	单词字符
[\b]	退格字符	\W	非单词字符
\c x	控制字符	\0	空字符
\d	数字字符	\x xx	字符的十六进制值
\D	非数字字符	\u xxx	字符的 Unicode 值
\o xxx	字符的八进制值		

2.5　匹配空白符

可以用以下简写式匹配空白符：

```
\s
```

请在 RegExr 试一试并看看高亮的部分（参见图 2-5）。以下字符组与 \s 匹配的内容相同：

```
[ \t\n\r]
```

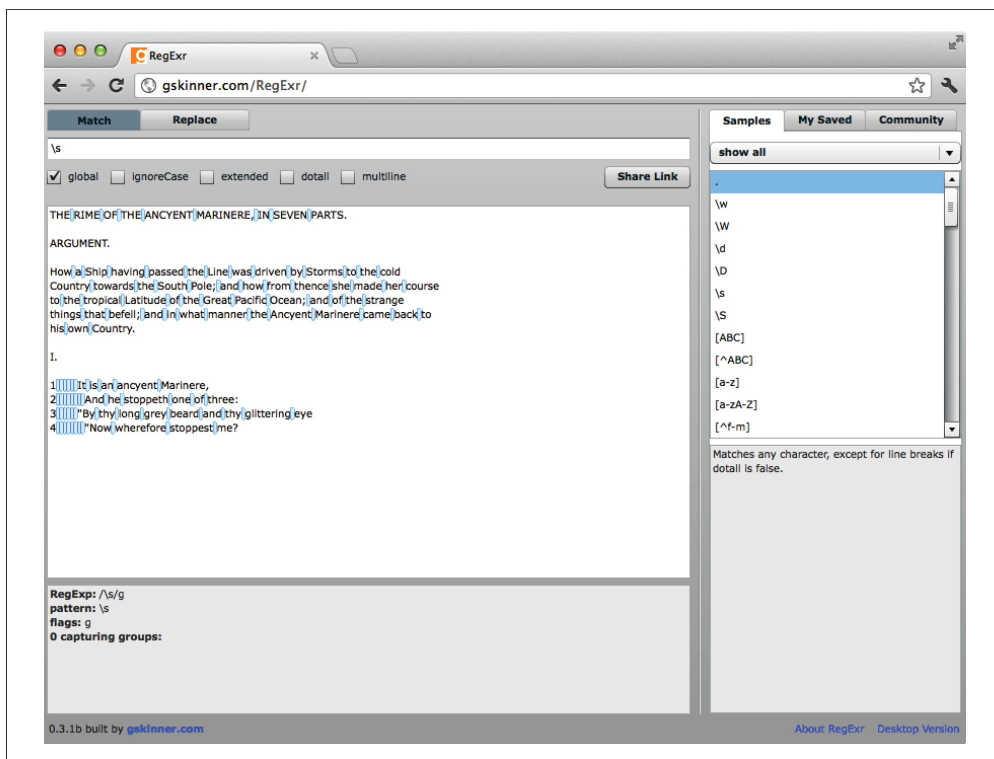


图 2-5：在 RegExr 中用 `\s` 匹配空白符

也就是说，它会匹配：

- 空格
- 制表符（`\t`）
- 换行符（`\n`）
- 回车符（`\r`）

在 RegExr 中空格符与制表符会被标亮，但换行符和回车符则不会。

可想而知，`\s` 也有对应的大写形式。要匹配非空白字符，则使用：

```
\S
```

这个简写式匹配除空白符之外的所有符号。它匹配字符组：

```
[^ \t\n\r]
```

或者是：

```
[^\s]
```

请在 RegExr 中试一下看看会发生什么。

除了 \s 匹配的字符之外，还有其他不太常见的空白字符。表 2-2 列出了匹配常见和不太常见的空白字符的简写式。

表2-2：匹配各种空白符的简写式

字符简写式	描　　述	字符简写式	描　　述
\f	换页符	\s	空白符
\h	水平空白符	\S	非空白符
\H	非水平空白符	\t	水平制表符
\n	换行符	\v	垂直制表符
\r	回车符	\V	非垂直制表符

2.6　再谈匹配任意字符

用正则表达式匹配任意字符的一种方法就是使用点号（U+002E）。点号可以匹配除行结束符之外的所有字符，个别情况除外。

在 RegExr 中，去掉对 global 复选框的勾选。这样，任何正则表达式都会匹配目标文本中第一个匹配项。

在 RegExr 上方的文本框中键入单个点号来匹配任意字符。

如图 2-6 所示，点号匹配了目标文本中的第一个字符 T。

要匹配 THE RIME 整个短语，则可使用八个点号：

```
........
```

但这种方法太麻烦，所以推荐用量词：

```
.{8}
```

这个表达式就能匹配前两个单词以及它们之间的空格，但只是*粗略地*匹配。勾选 global 旁的复选框，看看这个表达式还有什么作用，你就知道我所说的*粗略*是什么意思了。它匹配了连续多组的八个字符，头尾相连，只有目标文本的最后几个字符除外。

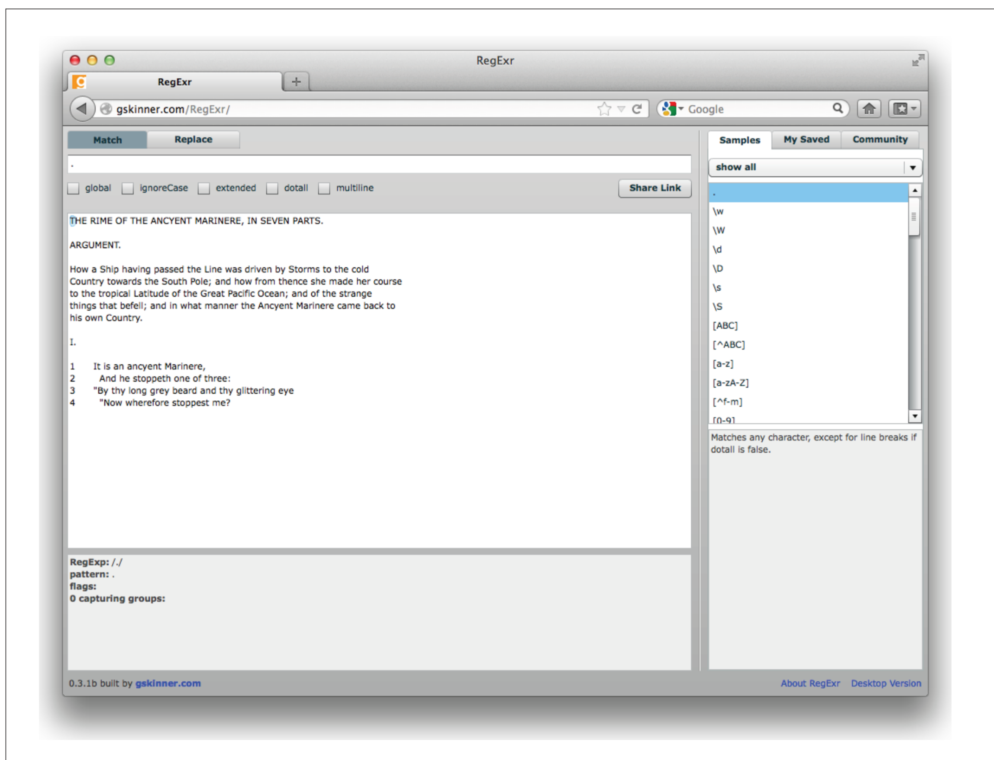


图 2-6：在 RegExr 中用 . 匹配单个字符

下面我们再试试匹配单词的边界和字母的开始和结束位置。在 RegExr 上方文本框中键入以下内容，可以看到细微的差异：

```
\bA.{5}T\b
```

这个表达式有更强的特指性（请记住*特指性*，specificity，这个概念很重要），它匹配单词 ANCYENT，也就是 ancient 的旧式拼写形式。这是如何做到的呢？

- 简写式 \b 匹配单词边界，不消耗任何字符；
- 字符 A 和 T 限定了字符序列的首尾字母；
- .{5} 匹配任意五个字符；
- 简写式 \b 匹配单词的另一个边界。

这个正则表达式实际上可以匹配 ANCYENT 和 ANCIENT。

现在再试一下这个简写式：

```
\b\w{7}\b
```

最后再试试匹配零个或多个字符：

```
.*
```

它就相当于：

```
[^\n]
```

或

```
[^\n\r]
```

类似地，点号也可以与表示“一个或多个”的量词（+）连用：

```
.+
```

请在 RegExr 中尝试这些表达式，它们都会匹配第一行内容（在取消勾选 global 复选框的情况下）。原因是点号通常不会匹配折行符，例如换行符（U+000A）或回车符（U+000D）。勾选 dotall 旁边的复选框，然后 `.*` 和 `.+` 就会匹配下方文本框中的全部文本。（dotall 表示点号匹配包括换行符在内的所有字符。）

这就涉及量词的贪心特性了。所谓“贪心”（greedy），指量词会匹配所有能匹配的字符。别急，第 7 章会详细介绍量词及其贪心特性。

2.7 给文本加标签

怎么让“The Rime of the Ancient Mariner”的内容以网页而不是纯文本形式显示呢？换句话说，怎么使用正则表达式而不是手写方式为它们添加 HTML5 标签呢？

在之后的几章中，我会陆续展示相应的方法，本章先从简单情况的开始。

点击 RegExr 中的 Replace（替换）标签，选中 multiline（多行）选项，然后在第一个文本框中键入：

```
(^T.*$)
```

这个表达式会从文件的开始匹配诗文的第一行，然后使用括号将文本捕获到一个分组中。在第二个文本框中键入：

```
<h1>$1<\h1>
```

这个替换表达式将 `$1` 捕获的内容嵌套在了 h1 标签中。可以在底部的文本框中看到结果。`$1` 是一个 Perl 风格的后向引用。此外，在包括 Perl 在内的多数实现程序中，还可以使用 `\1` 表示后向引用，但是 RegExr 只支持 `$1`、`$2`、`$3` 这样的方式。我们

会在第 4 章中讲解更多有关分组和后向引用的内容。

2.7.1　用sed为文本加标签

在命令行中使用 sed 也可以为文本添加标签。sed 是 Unix 流编辑器，它支持用正则表达式转换文本。sed 最初在 20 世纪 70 年代早期由 Lee McMahon 于贝尔实验室开发。如果用户使用的是 Mac 或者 Linux，那就已经有 sed 了。

请在 shell 提示符（比如 Mac 中的 Terminal 终端窗口）中测试以下内容：

```
echo Hello | sed s/Hello/Goodbye/
```

运行的过程应该如下：

- echo 命令将在标准输出设备（通常是屏幕）中打印单词 Hello，竖线符（|）将打印内容通过管道传到之后的 sed 命令；
- 管道将 echo 的输出转为 sed 的输入；
- sed 的 s 命令将单词 Hello 变为 Goodbye，而 Goodbye 就显示在屏幕上了。

如果你的平台上还没有安装 sed，可以参考本章末尾相关资源中的链接。其中还讨论了 BSD 和 GNU 两个版本的 sed。

现在试着在命令或 shell 提示符中键入：

```
sed -n 's/^/<h1>/;s/$/<\/h1>/p;q' rime.txt
```

而输出就是：

```
<h1>THE RIME OF THE ANCYENT MARINERE, IN SEVEN PARTS.</h1>
```

以下是正则表达式处理器的工作过程解析。

- 首先调用 sed 程序。
- sed 默认的操作是直接复制每行输入并输出，-n 选项覆盖了该默认操作。之所要覆盖默认操作，是因为我们只想让正则表达式影响第 1 行。
- s/^/<h1>/ 在行的开头（^）添加 <h1> 标签。
- 分号（;）用于分隔命令。
- s/$/<\/h1>/ 在行的结尾（$）添加 </h1> 标签。
- 命令 p 会打印受影响的那一行（第 1 行）。与 -n 不同，后者会打印所有行。
- 最后命令 q 会结束程序，这样 sed 程序就只会处理第 1 行。
- 所有的操作都是针对 rime.txt 文件执行的。

这行命令还有另一种写法，即用 -e 选项分别引导每个命令。我当然更喜欢使用带分号的写法，因为那种写法更简短。

```
sed -ne 's/^/<h1>/' -e 's/$/<\/h1>/p' -e 'q' rime.txt
```

可以将这些命令写到文件里，比如这里所示的文件 h1.sed（该文件就在之前所说的代码库里）：

```
#!/usr/bin/sed

s/^/<h1>/
s/$/<\/h1>/
q
```

若要运行该文件，请在与 rime.txt 同一个路径或文件夹里运行如下命令：

```
sed -f h1.sed rime.txt
```

2.7.2　用Perl为文本加标签

最后，我将展示如何用 Perl 来做类似的事。Perl 是由 Larry Wall 于 1987 年创立的一种通用程序设计语言。它以对正则表达式的强大支持和文本处理能力而闻名。

在命令提示符中键入

```
perl -v
```

然后回车，看看你的系统中是否已经安装了 Perl。该命令会返回系统中 Perl 的版本信息或者返回错误（参见 2.9 节）。

在命令提示符中键入：

```
perl -ne 'if ($. == 1) { s/^/<h1>/; s/$/<\/h1>/m; print; }' rime.txt
```

就可以得到和 sed 示例中一样的输出结果：

```
<h1>THE RIME OF THE ANCYENT MARINERE, IN SEVEN PARTS.</h1>
```

以下是这个 Perl 命令的执行过程解析。

- `perl` 调用了 Perl 程序。
- `-n` 选项输出全部输入内容（rime.txt 文件）。
- `-e` 选项允许在命令行（而不用在文件）中提交程序代码。
- `if` 语句检查是否在第一行。在 Perl 中 `$.` 是个特殊的变量，它匹配当前行。

- 第一个替换命令 s 先找到第一行的开头（^）然后插入 <h1> 标签。
- 第二个替换命令 s 搜寻行结尾（$）再插入 </h1> 标签。
- 替换命令最后的 m（多行）修饰符，表示将本行单独处理；这样，$ 就只匹配第一行的结尾而不会匹配整个文本的结尾。
- 最后在标准输出设备（屏幕）中打印出结果。
- 所有这些操作都是针对 rime.txt 文件的。

同样可以将这些命令放入程序文件中，比如示例代码库中的 h1.pl 文件。

```
#!/usr/bin/perl -n

if ($. == 1) {
  s/^/<h1>/;
  s/$/<\/h1>/m;
  print;
}
```

然后在 rime.txt 的同一个目录下，执行如下命令：

```
perl h1.pl rime.txt
```

用 Perl 语言实现相同任务的方式很多，也许这不是为文本添加标记的最有效的方法。很可能本书上市后，我又会想出使用 Perl（或其他工具）的更有效的方法。希望你也能够想到。

下一章我们将讨论边界和零宽度断言。

2.8 本章所学

- 如何匹配字符串字面值
- 如何匹配数字和非数字字符
- 什么是 global（全局）模式
- 字符简写式与字符组的比较
- 如何匹配单词与非单词字符
- 如何匹配空白字符
- 如何使用点号匹配任意字符
- 什么是 dotall 模式
- 如何使用 RegExr、sed 以及 Perl 在文本中插入 HTML 标记

2.9 相关资源

- 可以在 http://www.gskinner.com/RegExr 找到 RegExr 及其桌面版本（http://www.gskinner.com/RegExr/desktop/）。RegExr 内 嵌 在 Flex 3 中（http:// www.adobe.com/products/flex.html）且依赖于 ActionScript 正则表达式引擎（http://www.adobe.com/devnet/actionscript.html）。其使用的正则表达式与 JavaScript 中的相似（参见 https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/RegExp）。
- Git 是一个稳定的版本控制系统（http://git-scm.com）。GitHub 是 Git 项目的在线代码库（http://github.com）。如果你熟悉 Git 或者 Subversion、Mercurial 等最新的版本控制系统，我建议你使用 GitHub 代码库来获取本书的示例代码。
- HTML5（http://www.w3.org/TR/html5/）是 W3C HTML 的第 5 个主要修订版，是在万维网上发布内容所使用的标记语言。它处于草稿状态好几年了，且会定期修订，但它作为 HTML 4.01 和 XHTML 的后续版本已得到浏览器普遍支持。
- Unix/Linux 操作系统中一般都有 sed 程序，包括 Mac 系统（Darwin 或 BSD 版）。通过 Cygwin（http://www.cygwin.com）及类似的发布版，也可以在 Windows 上使用 sed，在 http://gnuwin32.sourceforge.net/packages/sed.htm 中还可以找到独立版本（当前版本为 4.2.1，参见 http://www.gnu.org/software/sed/manual/sed.html）。
- 你可能需要安装 Perl 才能运行本章的 Perl 代码。Mac OS X Lion 默认带有 Perl，多数 Linux 系统也是。如果你使用的是 Windows，则需要安装合适的 Cygwin 安装包(参见 http://www.cygwin.com)，或者从 ActiveState 网站下载最新的安装包(请访问 http://www.activestate.com/activeperl/downloads)。关于安装 Perl 的详细信息，请访问 http://learn.perl.org/installing/ 或者 http://www.perl.org/get.html。

要确定你的系统中是否已经有 Perl，可在 shell 提示符中输入以下命令。请先打开命令行或 shell 窗口，例如 Mac 中的 Terminal 窗口（在 Application/Utilities 路径下），或者是 Windows 的命令行窗口（打开开始菜单，在菜单栏底部的文本框中键入 cmd）。在命令提示符中键入：

```
perl -v
```

如果 Perl 已经安装好且能够运行，则这个命令会返回 Perl 的版本信息。笔者的 Mac 上运行的是 Lion 操作系统，已用编译源代码的方式（参见 http://www.cpan.org/src/5.0/perl-5.16.0.tar.gz）安装了最新版本的 Perl（5.16.0 是编写本书时最新的版本）。键入以上命令时，笔者得到了以下信息：

```
This is perl 5, version 16, subversion 0 (v5.16.0) built for darwin-2level

Copyright 1987-2012, Larry Wall
```

```
Perl may be copied only under the terms of either the Artistic License or the
GNU General Public License, which may be found in the Perl 5 source kit.

Complete documentation for Perl, including FAQ lists, should be found on
this system using "man perl" or "perldoc perl". If you have access to the
Internet, point your browser at http://www.perl.org/, the Perl Home Page.
```

在编译源代码和生成程序时，perl 和 perldoc 都安装在 /usr/local/bin 中，你需要将其加入你的系统路径中。关于设置路径变量的信息，请参见 http://java.com/en/download/help/path.xml。

第3章
边界

本章的重点是断言。断言标记边界，但是并不耗用字符。也就是说，字符并不会返回到结果中。断言也被称做零宽度断言（zero-width assertion）。零宽度断言不匹配字符，而是匹配字符串中的位置。其中的一些，比如 ^ 和 $，也叫做锚位符（anchor）。

本章讨论的边界有以下几种：

- 行或者字符串的起始与结束位置；
- 单词边界（两种）；
- 主题词的起始与结束位置；
- 引用字符串字面值的边界。

本章继续使用 RegExr，但这回我们变点儿花样，使用 Safari 浏览器（当然，你可以使用任何浏览器），如图 3-1 所示。目标文本还是 rime.txt 的前 12 行。在 Safari 浏览器中打开 http://gskinner.com/regexr，然后从代码库中将 rime.txt 的前 12 行复制到下方的文本框中。

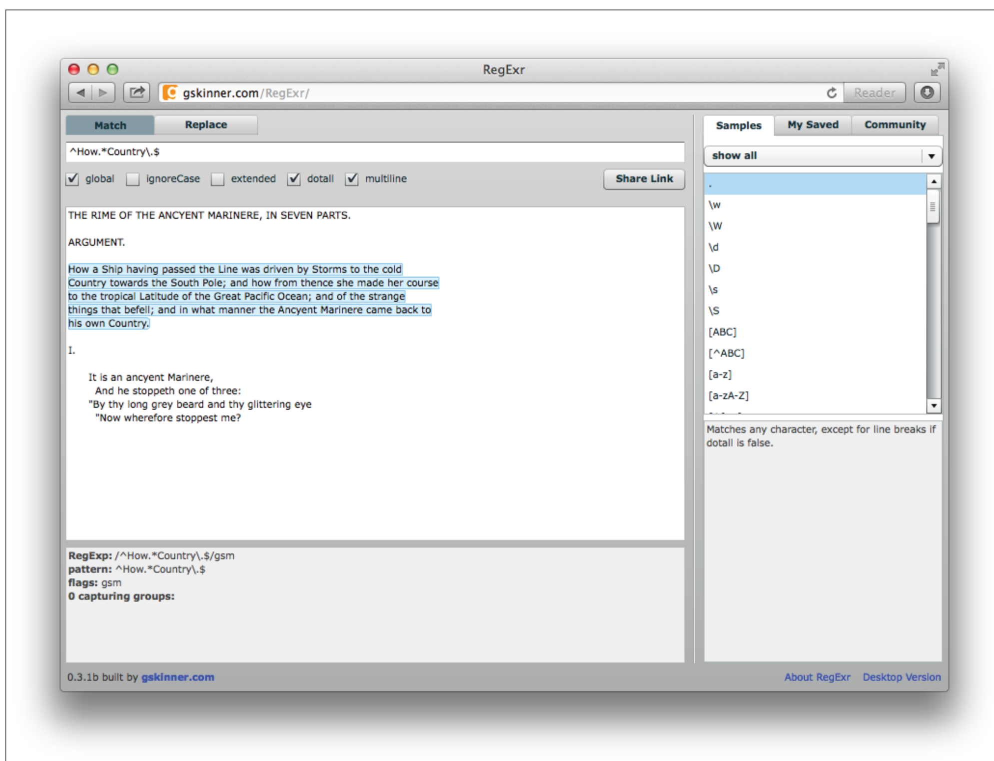


图 3-1：Safari 中的 RegExr

3.1 行的起始与结束

就像之前看到的那样，要匹配行或字符串的起始要使用脱字符（U+005E）：

```
^
```

根据上下文，^ 会匹配行或者字符串的起始位置，有时还会匹配整个文档的起始位置。而上下文则依赖于应用程序和在应用程序中所使用的选项。

如你所知，若要匹配行或字符串的结尾位置要使用美元符：

```
$
```

请确定 RegExr 中的 multiline（多行）选项被勾选。global（全局）选项在打开 RegExr 时默认勾选，但是本例中你可以勾选也可以不勾选该选项。如果不勾选 multiline 选项，整个目标文本被视做一个字符串。

在上方的文本框中键入下面这个正则表达式：

```
^How.*Country\.$
```

它会匹配以单词 How 开头的整行。请注意结尾的点号之前有一个反斜杠，它对点号进行转义，这样点号就被解释为字面值。如果不对点号转义，它就会匹配任意字符。如果想匹配作为字面值的点号，则必须将点号转义或者将其放入字符组中（参见第 5 章）。

如果不勾选 multiline 选项会是什么情况？高亮显示功能会被关闭。在不勾选 multiline 但选 dotall 的情况下，键入：

```
^THE.*\?$
```

可以看到它匹配了整个文本。

dotall 选项表示点号除了匹配其他字符之外，还会匹配换行符。取消勾选 dotall 选项，则该表达式什么也不匹配。而以下表达式：

```
^THE.*
```

则匹配第一行。再点击 dotall 选项，全部文本都会被匹配。不需要使用 `\?$` 来匹配文本的结尾。

3.2　单词边界与非单词边界

我们已经多次遇到使用 `\b` 的情况了。它匹配单词边界。请尝试：

```
\bTHE\b
```

（在勾选 global 选项的情况下）这个表达式会匹配第一行的两个 THE。就像 ^ 和 `$` 一样，`\b` 是个零宽度断言，表面上它会匹配空格或者是行起始，而实际上它匹配的是个零宽度的不存在的东西。你有没有注意到第二个 THE 两边的空格是没有标亮的？这是因为它们不是匹配的部分。这个理解起来不是很容易，但你可以通过观察它匹配和不匹配的内容来理解。

你还可以匹配非单词边界。非单词边界匹配除单词边界之外的位置，比如单词或者字符串中的字母或数字。要匹配一个非单词边界，试一下：

```
\Be\B
```

看看它匹配了什么（参见图 3-2）。可以看到它匹配了小写字母 e，而匹配的字母 e 的两边都是其他字母或者是非单词字符。零宽度断言不会匹配两边的字符，但它会识别文字 e 的两边是否是非单词边界。

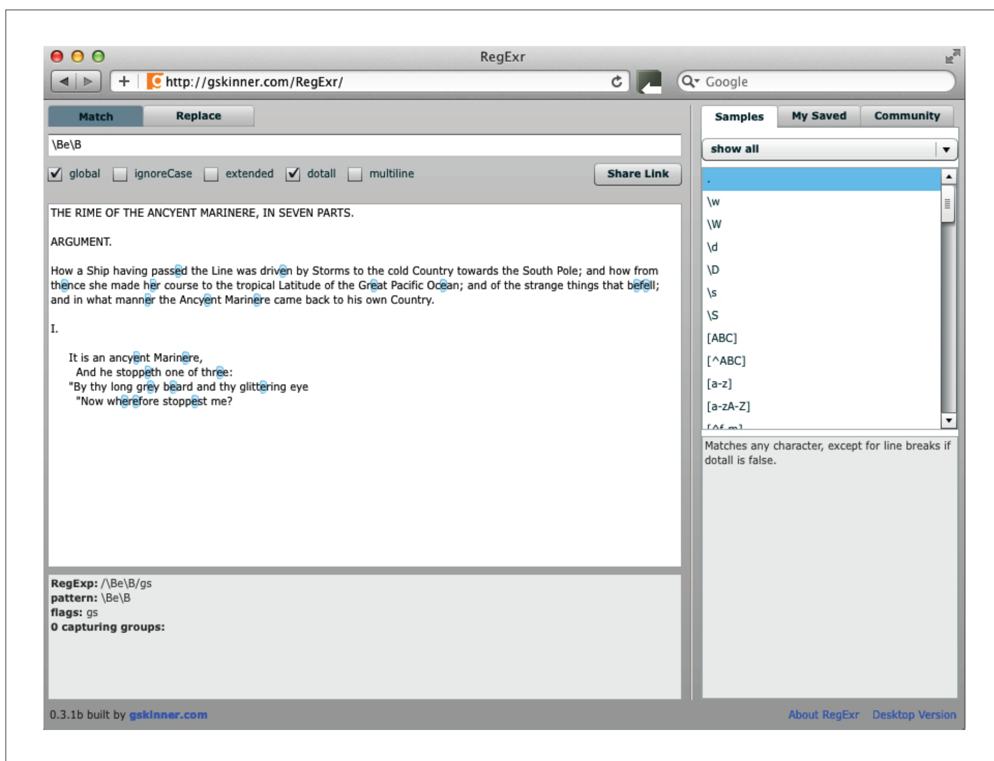


图 3-2：用 \B 来匹配非单词边界

在有些应用程序中，指定单词边界的另一种方法是使用：

```
\<
```

来指定单词的开头，而使用

```
\>
```

来指定单词结尾。这是比较旧的语法，在很多最新的正则表达式应用程序中无法使用。但在有些情况下，这种语法就很有用，因为它不会像 \b 那样匹配任意单词边界，而是允许你分别匹配单词的开头或结尾。

如果你的系统上有 vi 或者 vim，可以用这类编辑器试一下。即使你没用过 vim，这也很简单，只要按照以下步骤做即可。在命令行或者 shell 窗口中，将路径改为诗文所在的地方，再用这个命令打开：

```
vim rime.txt
```

然后输入以下查找命令：

```
/\>
```

再按回车键（Enter）或返回键（Return）。在 vim 中使用斜杠符（/）开始一次查找。请观察光标，可以看到本次查找过程会找到单词的结尾。按下按键 `n` 可重复查找。再输入：

```
/\<
```

再按回车键（Enter）或返回键（Return）。这次查找会找出单词的开头。要退出 vim，键入 `ZZ` 即可。

这一语法也可用于 grep。自从 20 世纪 70 年代，grep 像 sed 一样一直是 Unix 系统的主要工具。（20 世纪 80 年代，我一个同事的车牌上面就写着 GREP。）在 shell 提示符中试试这个命令：

```
grep -Eoc '\<(THE|The|the)\>' rime.txt
```

-E 选项表示要使用扩展的正则表达式（ERE），而不是 grep 默认使用的基本正则表达式（BRE）。`-o` 选项代表结果只显示一行中与模式相匹配的那部分。`-c` 选项的意思是只返回结果的数量。单引号中的模式会对 THE、The 或者 the 进行整词匹配。这就是 `\<` 和 `\>` 帮你寻找的。

这个命令返回的是：

```
259
```

也就是找到的单词的数目。

另一方面，如果不使用 `\<` 和 `\>`，结果则不同。这样做：

```
grep -Eoc '(THE|The|the)' rime.txt
```

得到的数字就是：

```
327
```

为什么？因为该模式只匹配整词和任意包含该词的字符序列。这就是 `\<` 和 `\>` 能派上用场的一个原因。

3.3　其他锚位符

与锚位符 `^` 相似，以下简写式匹配主题词的起始：

```
\A
```

这个写法不是在所有的正则表达式程序中都可以使用的，但可以在 Perl 和 PCRE 中使用。要匹配主题词的结尾，可以使用：

```
\Z
```

在某些上下文中还可用：

```
\z
```

pcregrep 是带有 PCRE 库的 grep 版本。（如何得到 pcregrep 请参见 3.7 节。）安装之后，要使用以上语法，则这样写：

```
pcregrep -c '\A\s*(THE|The|the)' rime.txt
```

单词 the 出现在行首附近位置且之前有（一个或多个）空格的次数为 108 次，命令 `-c` 会返回这个次数。接下来输入命令：

```
pcregrep -n '(MARINERE|Marinere)(.)?\Z' rime.txt
```

这一命令会匹配一行（主题词）尾部的 MARINERE 或 Marinere，之后是任何可选字符，在本例中可选字符就是标点符号或者字母 S。（点号两边的括号不是必需的。）

可以看到输出为：

```
1:THE RIME OF THE ANCYENT MARINERE,
10:     It is an ancyent Marinere,
38:       The bright-eyed Marinere.
63:       The bright-eyed Marinere.
105:     "God save thee, ancyent Marinere!
282:     "I fear thee, ancyent Marinere!
702:     He loves to talk with Marineres
```

pcregrep 的 `-n` 选项在输出的每行起始处显示行号。pcregrep 与 grep 的命令行选项十分相似。要了解所有选项，输入：

```
pcre --help
```

3.4　使用元字符的字面值

可以用

```
\Q
```

和

```
\E
```

之间的字符集匹配字符串字面值。

为了展示这一点，在 RegExr 下方文本框中输入以下元字符：

```
.^$*+?|(){}[]\-
```

这 15 个元字符在正则表达式中有特殊含义，用来编写匹配模式。（连字符在字符组的方括号中用来表示范围。但在其他情况下，则无特殊含义。）

如果你在 RegExr 上方的文本框中尝试匹配这些字符，则不会看到任何结果。为什么呢？因为 RegExr 会认为这是个正则表达式而不是字符串字面值。现在试一下

```
\Q$\E
```

它将匹配 `$`，因为 `\Q` 和 `\E` 之间的任意字符都会被解释为普通字符（参见图 3-3）。（记住，可以在元字符之前加一个 `\` 使其匹配字面值。）

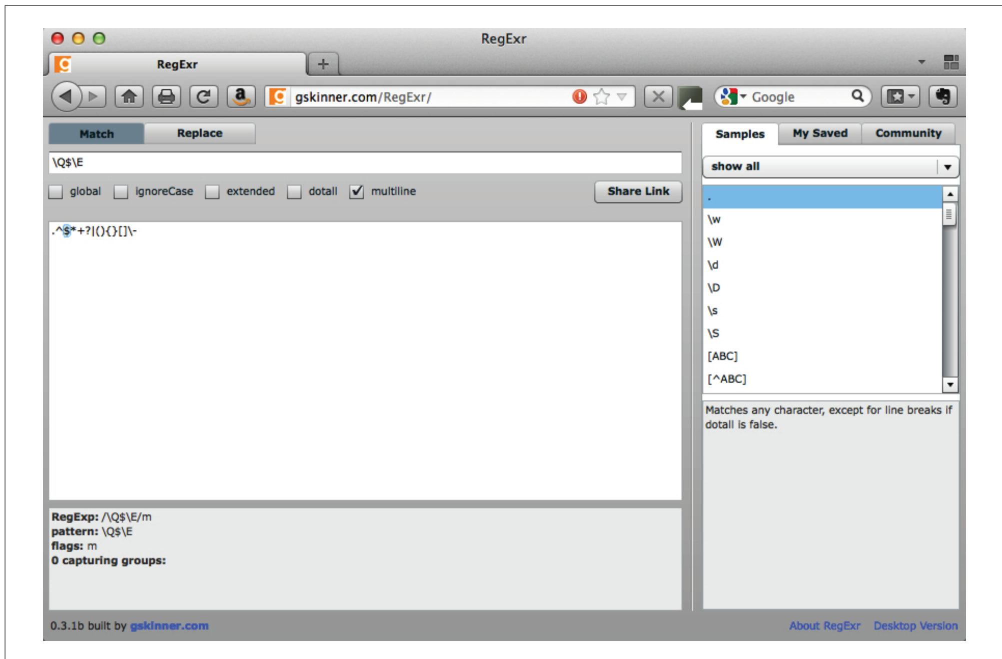


图 3-3：使用元字符的字面值

3.5　添加标签

在 RegExr 中，取消勾选 global 但勾选 multiline，点击 Replace 标签，然后在第一个文本框（在图 3-4 中标号为 1）中输入：

```
^(.*)$
```

这会匹配第一行文本并将其捕捉。然后在下一个文本框（标号为 2）中输入以下内容：

```
<!DOCTYPE html>\n<html lang="en">\n<head><title>Rime</title></head>\n
    <body>\n<h1>$1</h1>
```

输入替换文本的时候，你会注意（标号为 3 的文本框中的）主题词文本在显示结果的文本框（标号为 4）中变了，包含了你刚刚添加的标记（参见图 3-4）。

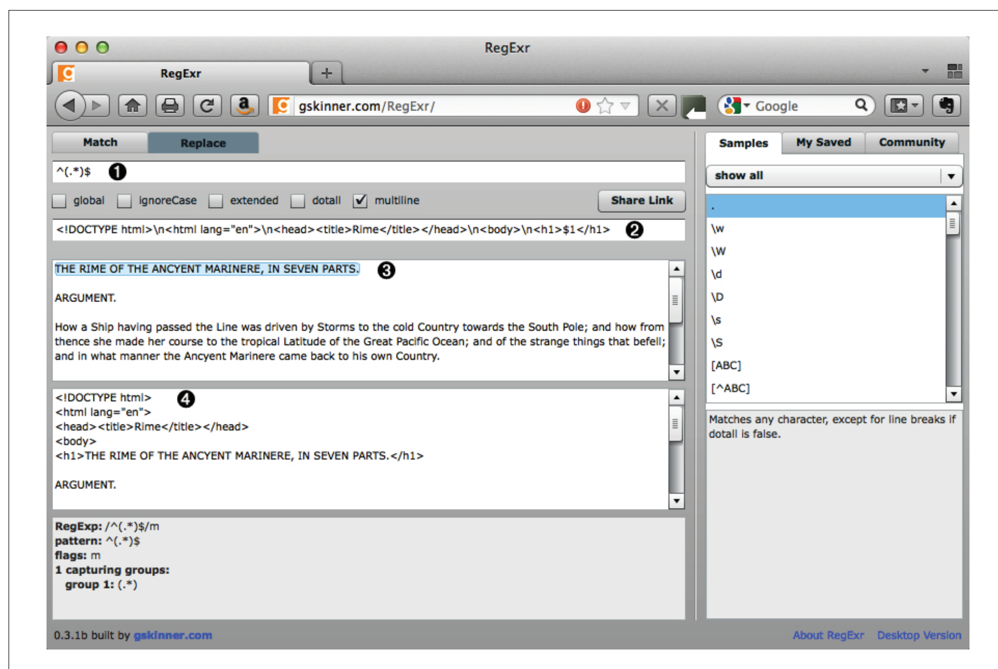


图 3-4：用 RegExr 添加标记

RegExr 很好地展示了添加标记的一种方式，但它也有自己的局限性，比如说它不能将结果保存为文件。因此，我们不能把目光仅局限于浏览器。

3.5.1 使用sed添加标签

第 2 章我们已经看到了，在 RegExr 中能完成的工作，完全可以使用 sed 在命令行环境下做到。sed 中的插入命令（`i`）允许你在文件或字符串中的某个位置之前插入文本。而与 `i` 命令相反的是命令 `a`，它在某个位置之后添加文本（后面会用到 `a`）。

以下命令从第 1 行开始插入 HTML5 的 doctype（文档类型）和其他标记：

```
sed '1 i\
<!DOCTYPE html>\
<html lang=\"en\">\
<head>\
<title>Rime</title>\
</head>\
<body>

s/^/<h1>/
s/$/<\/h1>/
q' rime.txt
```

行尾的反斜杠（\）允许你在该流中插入新行而不会提前执行命令。引号前的反斜杠将引号转义为字面值。

正确运行这个 sed 命令会得到如下输出：

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>The Rime of the Ancyent Mariner (1798)</title>
</head>
<body>
<h1>THE RIME OF THE ANCYENT MARINERE, IN SEVEN PARTS.</h1>
```

以上 sed 命令保存在实例代码库中的 top.sed 文件中。用以下命令运行这些代码：

```
sed -f top.sed rime.txt
```

可以得到与前面的命令相同的输出。要将输出保存到文件，则可以把输出重定位到一个文件中，比如：

```
sed -f top.sed rime.txt > temp
```

除了会在屏幕上显示结果，重定位的那部分（>temp）还会将输出保存到文件 temp 中。

3.5.2 使用Perl添加标签

下面尝试用 Perl 完成同样的工作。先不解释，试一试这个：

```
perl -ne 'print "<!DOCTYPE html>\
<html lang=\"en\">\
<head><title>Rime</title></head>\
<body>\
" if $. == 1;
s/^/<h1>/;s/$/<\/h1>/m;print;exit;' rime.txt
```

与之前的 sed 命令比较一下。有哪些地方相似，哪些地方不同？ sed 命令稍微简单一些，但笔者认为 Perl 的功能强大很多。

以下是该命令的工作过程。

- 变量 $. 表示当前行，使用 if 语句测试。如果 if 语句返回 true（真），则当前行就是第 1 行。
- 当 Perl 用 if 语句找到第 1 行时，它会打印文档类型（doctype）和几个 HTML 标签。和在 sed 中一样，这里也需要将引号转义。
- 第一个替换操作在行起始位置插入 <h1> 标签，第二个替换操作在行尾处插入 </h1> 标签。第二个替换操作结尾处的 m 表示使用多行修饰符。这就保证了该命令会识别第一行的结尾。如果没有 m，则 $ 会匹配该文件的结尾。
- 命令 print 打印替换结果。
- 命令 exit 则会立即结束 Perl 程序。否则，因为有 -n 选项，它会循环执行文件的每一行；但这里不需要。

以上命令要键入很多内容。我将这些 Perl 代码放入名为 top.pl 的文件中，也可以在代码库中找到。

```
#!/usr/bin/perl -n

if ($ == 1) {
print "<!DOCTYPE html>\
<html lang=\"en\">\
<head>\
<title>The Rime of the Ancyent Mariner (1798)</title>\
</head>\
<body>\
";
s/^/<h1>/;
s/$/<\/h1>/m;
print;
exit;
}
```

使用合集运行这个文件：

```
perl top.pl rime.txt
```

得到的结果与前面是一样的，只是形式上稍有不同。（就像在 sed 中一样，你可以用 > 将输出重定位到文件中。）

下一章会涉及选择、分组和后向引用，还有其他一些概念。第 4 章见！

3.6 本章所学

- 如何使用锚位符 ^ 或 $ 匹配行首或行尾

- 如何匹配单词或非单词边界
- 如何用 \A 和 \Z（或 \z）匹配主题词的起始与结束位置
- 如何用 \Q 和 \E 将字符串标引为字面值
- 如何用 RegExr、sed 和 Perl 为文本加标签

3.7 相关资源

- vi 是一个支持正则表达式的 Unix 编辑器，它于 1976 年由 Sun 公司的联合创始人 Bill Joy 开发。vim 编辑器最初由 Bram Moolenaar 开发（见 http://www.vim.org），它取代了 vi。Bill Joy 和 Mark Horton 执笔的一篇关于 vi 的早期论文可以在这里找到：http://docs.freebsd.org/44doc/usd/12.vi/paper.html。自从 1983 年第一次使用 vi 开始，我几乎每天都在使用它。相比于其他文本编辑器，使用 vi 可以事半功倍。它功能十分强大，虽然使用它已经近 30 年了，但我经常还会发现一些新功能。
- grep 是一个使用正则表达式来查找和打印字符串的 Unix 命令行工具。grep 由 Ken Thompson 于 1973 年发明，它被认为是从 ed 编辑器的 g/re/p（global/regular expression/print）命令衍生而来。后来又出现了功能更强大的 egrep（也称 grep-E），但 grep 没有被淘汰。egrep 使用了扩展的正则表达式（ERE）并引入了更多的元字符，比如 |、+、?、(、)。fgrep（grep-F）使用字符串字面值来查找文件；像 $、*、| 这样的元字符则没有特殊含义。在 Linux 系统和 Mac OS X 的 Darwin 系统中都可以使用 grep。也可以从 Cygwin 的 GNU 发布版（http://www.cygwin.com）中获得 grep，或者从 http://gnuwin32.sourceforge.net/packages/grep.htm 下载。
- PCRE（http://www.pcre.org）英文全称 Perl Compatible Regular Expression，是一个与 Perl 5 兼容的正则表达式的 C 函数库（8 位和 16 位），也包含其他实现的一些特性。pcregrep 是一个 8 位的类 grep 工具，它允许你在命令行中使用 PCRE 库的功能。运行命令 `sudo port install pcre`，即可通过 Macports（http://www.macports.org）为 Mac 安装 pcregrep。（前提是安装了 Xcode，见 https://developer.apple.com/technologies/tools/。需要注册。）

第4章

选择、分组和后向引用

我们已经在前面的示例中见过分组了。分组对文本加括号以帮助执行某种操作，比如：

- 在两种或更多可选模式中选择一个；
- 创建子模式；
- 捕获一个分组以便之后进行后向引用；
- 对组合的模式使用某项操作（如量词）；
- 使用非捕获分组；
- 原子分组（高级）。

本章我们会依旧使用 rime.txt 中“The Rime of the Ancyent Mariner”的文本，除此之外还会使用一些专门设计的例子。但这一次我们要使用 RegExr 的桌面版程序和 sed 等其他工具。可以从 http://www.regexr.com 下载 RegExr 的 Windows、Mac 和 Linux 版的桌面程序（用 Adobe Air 编写的）。在 RegExr 的网页中，点击页面右下角的 Desktop Version（桌面版）链接来获取更多信息。

4.1 选择操作

简单地说，选择操作可在多个可选模式中匹配一个。例如，你想在“The Rime of the Ancyent Mariner”中找出 the 出现过多少次，包括 THE、The 和 the 等形式。为此，就可以使用选择操作。

双击图标打开 RegExr 桌面应用程序。它看起来很像在线版本，但好处是在你的机器本地运行，因此上不了网也没关系。

我已将整首诗复制粘贴到 RegExr 的桌面版中供之后的练习使用。我是在运行 OS X Lion 的 Mac 上使用 RegExr 桌面版的。

在顶部的文本框中输入：

```
(the|The|THE)
```

你会看到下方文本框里诗文中所有的 the 都被标亮（参见图 4-1）。请使用滚动条查看更多的结果。

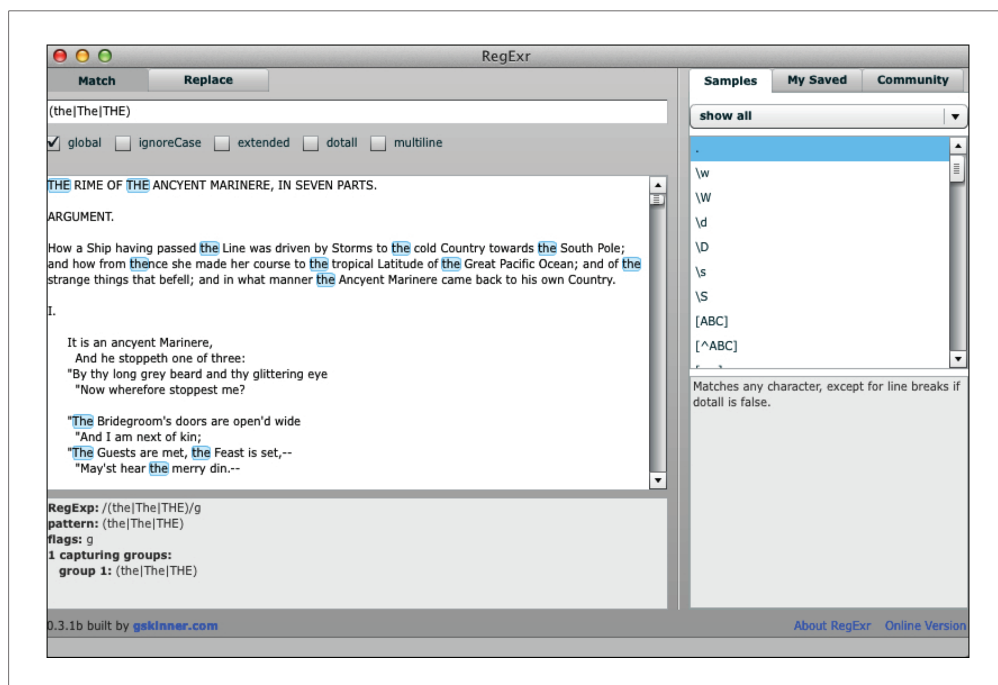


图 4-1：在 RegExr 桌面版中使用选择操作

可以使用一个选项来使分组更简短。借助选项，可以指定查找模式的方式。例如，选项

```
(?i)
```

让你的模式不再区分大小写。因此原来带选择操作的模式可以简写成

```
(?i)the
```

在 RegExr 中试一下，看看效果如何。还可以在 RegExr 中勾选 ignoreCase 来设定不区分字母大小写，这两种方式都可以。表 4-1 中列出了以上选项以及其他各种选项和修饰符。

表4-1：正则表达式中的选项

选　　项	描　　述	支持平台
`(?d)`	Unix 中的行	Java
`(?i)`	不区分大小写	PCRE、Perl、Java
`(?J)`	允许重复的名字	PCRE*
`(?m)`	多行	PCRE、Perl、Java
`(?s)`	单行（dotall）	PCRE、Perl、Java
`(?u)`	Unicode	Java
`(?U)`	默认最短匹配	PCRE
`(?x)`	忽略空格和注释	PCRE、Perl、Java
`(?-…)`	复原或关闭选项	PCRE

* 参见 http://www.pcre.org/pcre.txt 中的“Named Subpatterns”（命名子模式）。

接下来我们在 grep 中使用选择操作。表 4-1 中的选项并不适用于 grep，所以你要使用原来的选择模式。要对单词 the 出现一次或多次的行的数目进行统计，且不区分大小写，则用：

```
grep -Ec "(the|The|THE)" rime.txt
```

可以得到这个结果：

```
327
```

这个结果还不能说明一切。欲知原因，请继续阅读。

以下是对上面的 grep 命令的分析。

- `-E` 选项表示使用扩展的正则表达式（ERE），而不用基本正则表达式（BRE）。本例省去了将括号和竖线符转义的步骤，如果使用 BRE 则必须进行转义，像这样 `\(THE\|The\|the\)`。
- `-c` 选项返回匹配的行数（不是匹配的单词）。
- 括号将对 the、The 以及 THE 的选择操作归为一个分组。
- 竖线符将备选项分开，对备选项求值的顺序是从左至右。

下面的方法会将单词的每一次出现都作为一行内容返回，这样就可以得到实际的出现次数：

```
grep -Eo "(the|The|THE)" rime.txt | wc -l
```

这个命令返回的是：

```
412
```

以下是对命令的分析。

- `-o` 选项表示只显示一行中与指定模式匹配的部分，而由于导向 `wc` 的管道（`|`），该选项的作用不太直观。
- 在此上下文中，竖线符将 grep 命令的输出通过管道导向 `wc` 命令的输入。`wc` 是单词计数命令，而 `-l` 对输入的行数进行统计。

为什么有 327 与 412 这样大的差异呢？因为 `-c` 给出的是匹配的行的数目，但是一行中可能有多个单词匹配。若使用 `-o` 和 `wc -l`，则指定单词不管以哪一种形式出现，每次出现都会作为单独的一行统计，这样结果值就会更大。

要使用 Perl 执行同种匹配，则这样写：

```
perl -ne 'print if /(the|The|THE)/' rime.txt
```

或者更好的方式是不用选择操作，而使用前面提到的 `(?i)` 选项：

```
perl -ne 'print if /(?i)the/' rime.txt
```

再进一步改善，就是将 `i` 修饰符加在定界符之后：

```
perl -ne 'print if /the/i' rime.txt
```

结果相同，但命令是越简单越好。表 4-2 中列出了更多的修饰符（也称为标记符）。请与表 4-1 中的选项（两者相似但语法不同）进行比较。

表4-2：Perl语言中的修饰符（标记符）*

修饰符	描　　述
`a`	匹配 `\d`、`\s`、`\w` 以及处于 ASCII 范围内的 POSIX 字符
`c`	匹配失败后则停留在当前位置
`d`	使用默认的本地平台的规则
`g`	全局匹配
`i`	匹配时不区分大小写
`l`	使用当前位置的规则
`m`	多行字符串
`p`	保留匹配的字符串
`s`	将字符串看做一行内容
`u`	匹配时使用 Unicode 规则
`x`	忽略空格及注释

* 参见 http://perldoc.perl.org/perlre.html#Modifiers。

4.2　子模式

多数情况下，提到正则表达式中的*子模式*（subpattern），就是指分组中的一个或多个分组。子模式就是模式中的模式。多数情况下，子模式中的条件能得到匹配的前提是前面的模式得到匹配，但也有例外。子模式的写法可以有很多种，这里我们主要关注括号中的子模式。

从某种意义上说，你之前所见的模式：

```
(the|The|THE)
```

有三个子模式：the 是第一个子模式，The 是第二个，而 THE 是第三个。但是这种情况下，匹配第二个子模式不依赖于是否匹配第一个。（最左边的模式会首先匹配。）

而在以下的模式中，子模式依赖于前面的模式：

```
(t|T)h(e|eir)
```

通俗地讲，这个模式会匹配字面值 t 或 T，然后是一个 h，接下来就是一个 e 或者是 eir。相应地，这个模式会匹配以下所有情况：

- the；
- The；
- their；
- Their。

在以上情况中，第二个子模式 `(e|eir)` 依赖于第一个子模式 `(tT)`。

括号对于子模式不是必需的。下面是一个使用字符组的子模式示例：

```
\b[tT]h[ceinry]*\b
```

这个模式会匹配 the 或 The 还有 thee、thy 以及 thence 等单词。两个单词边界（`\b`）表示该模式只匹配整个单词，而不会匹配单词中的某几个字母。

以下是对这个模式的解析。

- `\b` 匹配单词起始边界。
- `[tT]` 是字符组，它匹配小写字母 t 或者大写字母 T。可以将其看做是第一个子模式。
- 然后匹配（或尝试匹配）小写字母 h。
- 第二个也就是最后一个子模式也表示为字符组 `[ceinry]`，其后用量词 `*` 表示零个或多个。

- 最后，该模式以另外一个 \b 结束。



正则表达式的术语经常是含义相近但使用范围迥异。为了在本书中解释子模式和其他概念，我查阅了很多资料并试着将它们的表述统一起来。但肯定会有人争辩说字符组不是子模式。我的观点是它们与子模式起到的作用一样，所以两者可以归为一类。

4.3 捕获分组和后向引用

当一个模式的全部或者部分内容由一对括号分组时，它就对内容进行捕获并临时存储于内存中。可以通过后向引用重用捕获的内容，形式为：

```
\1
```

或

```
$1
```

这里 \1 或 $1 引用的是第一个捕获的分组，而 \2 或 $2 引用第二个捕获的分组，以此类推。sed 只接受 \1 这种形式，而 Perl 则两种都接受。



最开始 sed 支持范围从 \1 到 \9 的后向引用，但现在已经没有这样的限制了。

下面再展示一下后向引用的使用方法。我们将使用它来重新排序诗文中的一行词，在此先跟作者柯勒律治道个歉。点击 RegExr 的 Repalce 标签，在顶部的文本框中输入下面的模式：

```
(It is) (an ancyent Marinere)
```

向下滚动主文本框（第三个文本区）直到你可以看到被标亮的那一行，然后在第二个文本框中，输入：

```
$2 $1
```

就可以看到在最下面的文本框中那一行被重新排列为：

```
an ancyent Marinere It is,
```

结果参见图 4-2。

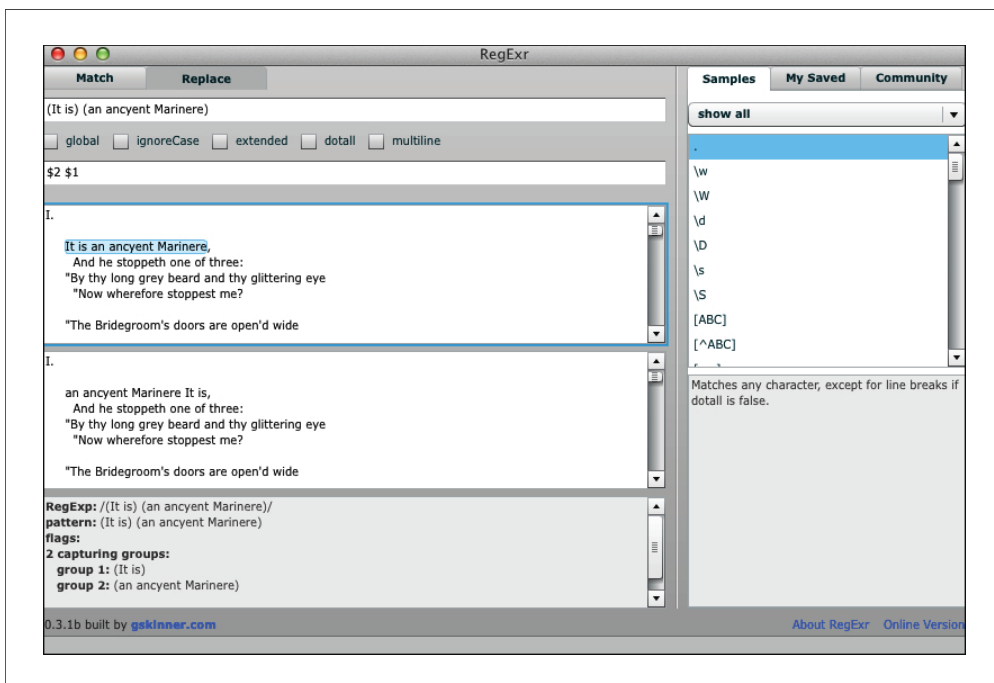


图 4-2：用 `$1` 和 `$2` 进行后向引用

要用 sed 得到相同结果，可以这样做：

```
sed -En 's/(It is) (an ancyent Marinere)/\2 \1/p' rime.txt
```

输出为：

```
an ancyent Marinere It is,
```

和 RegExr 中一样。为了帮助你理解每个细节，下面我们来分析一下这个 sed 命令。

- `-E` 选项仍然是调用 ERE（扩展的正则表达式），因此，括号可以直接当成字面值来使用了。
- `-n` 选项覆盖打印每一行的默认设置。
- 替换命令搜索与文本“It is an ancyent Marinere,”匹配的内容，再将其捕获放入两个分组中。
- 替换命令还将捕获的文本重排为先是后向引用 \2 的内容再是 \1 的内容，再将匹配的文本替换为重排后的内容并输出。
- 替换命令结尾处的 `p` 表示要打印该行。

Perl 中类似的命令会做相同的事：

```
perl -ne 'print if s/(It is) (an ancyent Marinere)/\2 \1/' rime.txt
```

注意该命令使用了 `\1` 风格的语法。当然，你也可以使用 `$1` 语法：

```
perl -ne 'print if s/(It is) (an ancyent Marinere)/$2 $1/' rime.txt
```

我喜欢 Perl 允许不必在循环中跳转就能打印指定行的方式。

但关于输出，还有一点要注意的是：

```
an ancyent Marinere It is,
```

在转换过程中某些单词首字母的大小写被打乱了。Perl 可以使用 `\u` 和 `\l` 来修正这个问题，就是这样：

```
perl -ne 'print if s/(It is) (an ancyent Marinere)/\u$2 \l$1/' rime.txt
```

现在结果看起来好多了：

```
An ancyent Marinere it is,
```

接下来解释一下原因。

- `\l` 语法并不匹配任何字符，而是会将紧接其后的字母变为小写。
- `\u` 语法将紧接其后的字母变为大写。
- `\U` 指令（未展示）将紧接其后的文本字符串全部变为大写。
- `\L` 指令（未展示）将紧接其后的文本字符串全部变为小写。

这些指令在文本中出现其他指令（比如 `\l` 或 `\E`，作为文字使用的字符串的结尾）之前都是起作用的。请自己动手试试这些指令。

命名分组

命名分组（named group）就是有名字的分组。这样，就可以通过名字（而不是数字）来引用分组。下面展示一下 Perl 语言中如何使用命名分组：

```
perl -ne 'print if s/(?<one>It is) (?<two>an ancyent Marinere)/\u$+{two}
       \l$+{one}/' rime.txt
```

我们来看看这个命令：

- 在括号内添加 `?<one>` 和 `?<two>` 将分组分别命名为 one 和 two；
- `$+{one}` 引用名为 one 的分组，而 `$+{two}` 则引用名为 two 的分组。

如果在一个模式中有分组被命名，那么你还可以重用该命名分组。解释一下这句话

的意思，假设你要查找含有连续六个 0 的字符串：

```
000000
```

这个例子很浅显，但可以说明其工作原理。用这一模式对连续三个 0 的分组命名（其中 z 是分组名，可以任意取）：

```
(?<z>0{3})
```

然后你可以再使用该分组，就像这样：

```
(?<z>0{3})\k<z>
```

或者：

```
(?<z>0{3})\k'z'
```

或者：

```
(?<z>0{3})\g{z}
```

请在 RegExr 中尝试一下看看结果。这些示例应该都能用。表 4-3 列出了命名分组可能使用的各种语法。

表4-3：命名分组的语法

语　法	描　述
`(?<name>…)`	命名分组
`(?name…)`	另一种命名分组的方式
`(?P<name>…)`	Python 中的命名分组
`\k<name>`	在 Perl 中引用分组名
`\k'name'`	在 Perl 中引用分组名
`\g{name}`	在 Perl 中引用分组名
`\k{name}`	在 .NET 中中引用分组名
`(?P=name)`	在 Python 中中引用分组名

4.4　非捕获分组

还有一种分组是*非捕获分组*（Non-Capturing Group）。非捕获分组不会将其内容存储在内存中。在你并不想引用分组的时候，可以使用它。由于不存储内容，非捕获分组就会带来较高的性能，而运行本书的简单示例很难察觉到性能的提升。

还记得本章中讨论的第一个分组吗？就是这个：

```
(the|The|THE)
```

你不需要任何后向引用，因此可以这样写一个非捕获分组：

```
(?:the|The|THE)
```

回到本章开头，你可以添加选项将其变为不区分大小写的模式，就像这样：

```
(?i)(?:the)
```

或者你也可以这样做：

```
(?:(?i)the)
```

不过，下面这种写法最值得推荐：

```
(?i:the)
```

该选项 `i` 可以放在问号和冒号之间。

原子分组

另一种非捕获分组是*原子分组*（atomic group）。如果你使用的正则表达式引擎进行回溯操作，这种分组就可以将回溯操作关闭，但它只针对原子分组内的部分，而不针对整个正则表达式。其语法如下：

```
(?>the)
```

什么时候你会想使用原子分组呢？正则表达式处理过程缓慢的一个因素就是回溯操作。其原因就是回溯操作会尝试每一种可能性，这会消耗时间和计算资源。有时它会占用大量时间。回溯有可能产生巨大的负面效应，这被称为*灾难性回溯*。

使用像 re2（http://code.google.com/p/re2/）这样的非回溯引擎可彻底关闭回溯操作，而使用原子分组可以关闭正则表达式的部分回溯操作。



本书的重点是介绍语法，不讨论性能调优问题。在我看来，原子分组主要与性能相关。

接下来第 5 章我们学习字符组。

4.5 本章所学

- 使用选择操作对两个或更多的模式进行选择

- 什么是选项及其修饰符，如何在模式中使用它们
- 各种类型的子模式
- 如何使用捕获分组和后向引用
- 如何使用命名分组，如何引用它
- 如何使用非捕获分组
- 关于原子分组的简单介绍

4.6 相关资源

- Adobe AIR 运行时程序允许你使用 HTML、JavaScript、Flash 以及 ActionScript 来构建 Web 应用程序，这种应用程序是独立的客户端应用，不需要使用浏览器。更多信息请查看 http://www.adobe.com/products/air.html。
- Python（http://www.python.org）是个很容易理解的高级程序设计语言。它有一个正则表达式实现（参见 http://docs.python.org/library/re.html）。
- .NET（http://www.microsoft.com/net）是一个 Windows 平台的编程框架。它也有正则表达式实现（http://msdn.microsoft.com/en-us/library/hs600312.aspx）。
- 更多有关原子分组的说明可在 http://www.regular-expressions.info/atomic.html 和 http://stackoverflow.com/questions/6488944/atomic-group-and-non-capturing-group 中找到。

第5章

字符组

这一章，我们详细讨论一下字符组。字符组有时也被称为方括号表达式（bracketed expression）。字符组有助于匹配特定字符或者特定的字符序列。它们可以像字符简写式那样代表一大批字符。比如，\d 匹配的字符与

```
[0-9]
```

所匹配的字符一样。

但字符组更有针对性，因此用途比简写式更广。

请在自己使用的正则表达式处理器中尝试以下示例。笔者在 Opera 中使用 Rubular，而在桌面使用 Reggy。

要做这个试验，请在网页的主文本区或者目标区域输入以下字符串：

```
! " # $ % & ' ( ) * + , - . /
0       1       2       3       4       5       6       7       8       9
: ; < = > ? @
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
[ \ ] ^ _ `
a b c d e f g h i j k l m n o p q r s t u v w x y z
{ | } ~
```

没有必要亲自动手输入这些字符，在本书配套代码库中的 ascii-graphic.txt 里能找到该文本。

请先使用字符组来匹配一个英文字符集。本例中，我们匹配英文元音字母：

```
[aeiou]
```

在下方文本框中，小写的元音字母应该被标亮了（见图 5-1）。如何将大写元音字母标亮？如何同时匹配或标亮小写和大写元音字母？

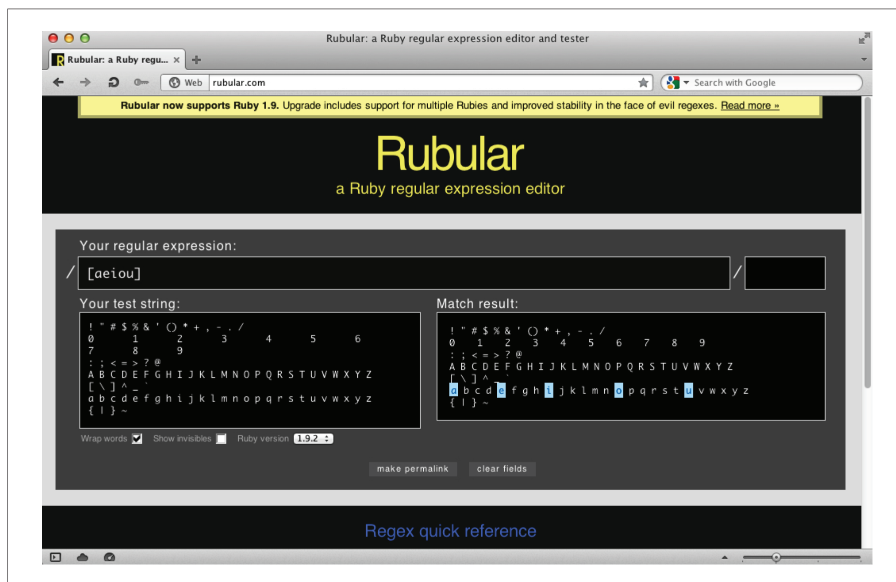


图 5-1：在 Opera 浏览器中使用 Rubular 的字符组功能

还可以使用字符组匹配某个范围的字符，比如

```
[a-z]
```

匹配的是从 a 到 z 的小写字母。试一下匹配更小范围的字符，比如从 a 到 f：

```
[a-f]
```

当然你还可以指定一个范围的数字：

```
[0-9]
```

或者让范围更小：

```
[3-6]
```

现在发挥一下想象力，如果你想匹配 10~19 的偶数，可以将两个字符组合起来写，像这样：

```
\b[1][24680]\b
```

同理，也能想到用下面的表达式查找 0~99 的偶数：

```
\b[24680]\b|\b[1-9][24680]\b
```

要想创建一个匹配十六进制数的字符组怎么做呢？这里给出一个提示：

```
[a-fA-F0-9]
```

还可以在字符组内使用简写式。例如，要匹配空格和单词字符，可以创建这样一个字符组：

```
[\w\s]
```

它等同于：

```
[_a-zA-Z \t\n\r]
```

但是更容易键入。

5.1 字符组取反

对字符组取反的语法之前你已经见过好几次了，这里我就简单介绍一下。对字符组取反会匹配与字符组内容不匹配的字符。比如，如果你不想匹配元音字符，可以这样写（请在浏览器中尝试，再参见图 5-2）：

```
[^aeiou]
```

该字符组起始位置的脱字符（^）的意义就是：“不，我不想匹配这些字符。”（脱字符必须出现在起始位置。）

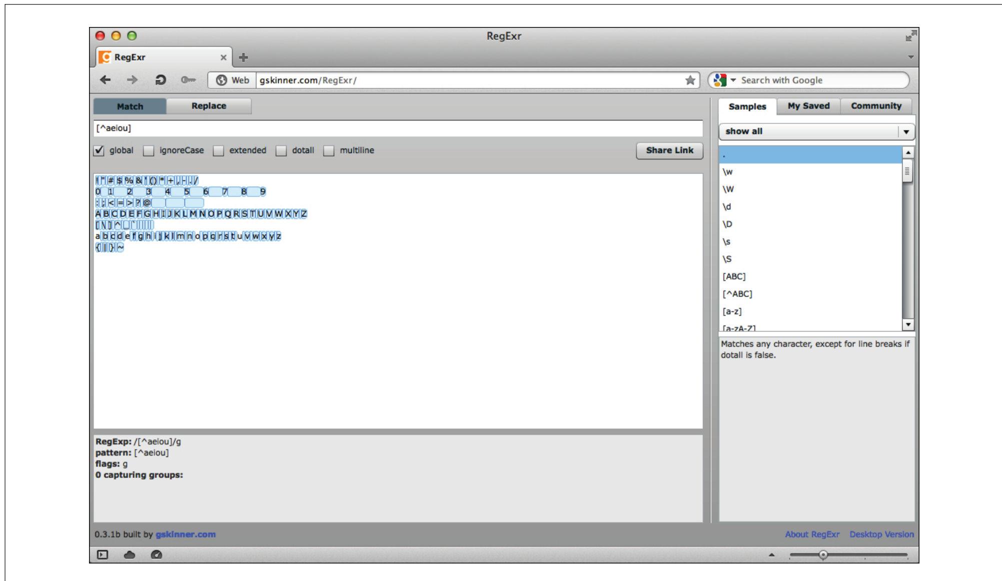


图 5-2：在 Opera 浏览器中使用 RegExr 测试对字符组取反

5.2　并集与差集

字符组可以像集合那样操作。事实上，字符组的另一个名称就是字符集（character set）。不是所有的实现程序都支持这项功能，但 Java 支持该功能。

我将展示一个叫做 Reggy 的 Mac 桌面应用程序（参见 5.5 节）。在其 Preference（偏好设置）面板中（如图 5-3 所示），我将 Regular Expression Syntax（正则表达式语法）改为“Java”，而在 Font 中（在 Format 里），为了阅读方便，我将字体大小设为 24。

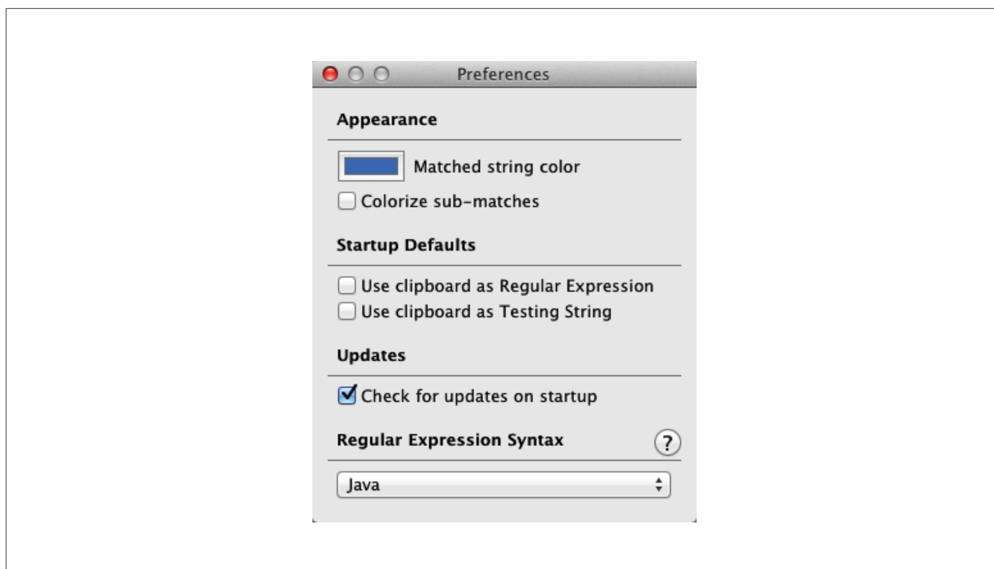


图 5-3：Reggy 的偏好设置

如果你想要两个字符组的并集，可以这样做：

```
[0-3[6-9]]
```

正则表达式处理器会匹配 0 到 3 之间的数字或者 6 到 9 之间的数字。图 5-4 展示了这在 Reggy 中的结果。

匹配差集（实质上就是减操作）：

```
[a-z&&[^m-r]]
```

这匹配 a 到 z 之间的字符，但其中 m 到 r 之间的字符除外（如图 5-5 所示）。

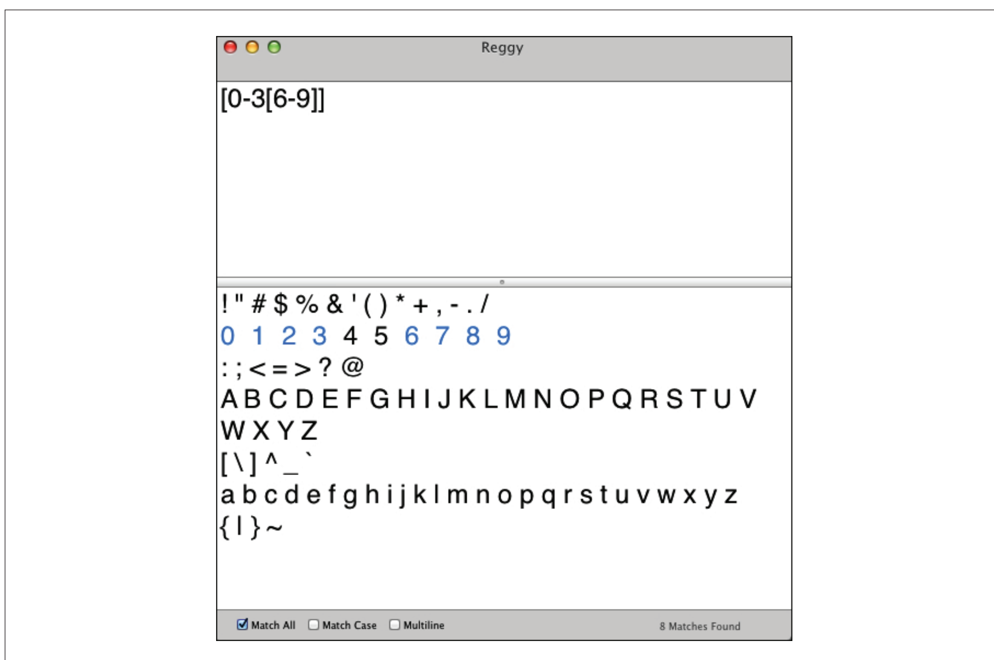


图 5-4：Reggy 中两个字符组的并集

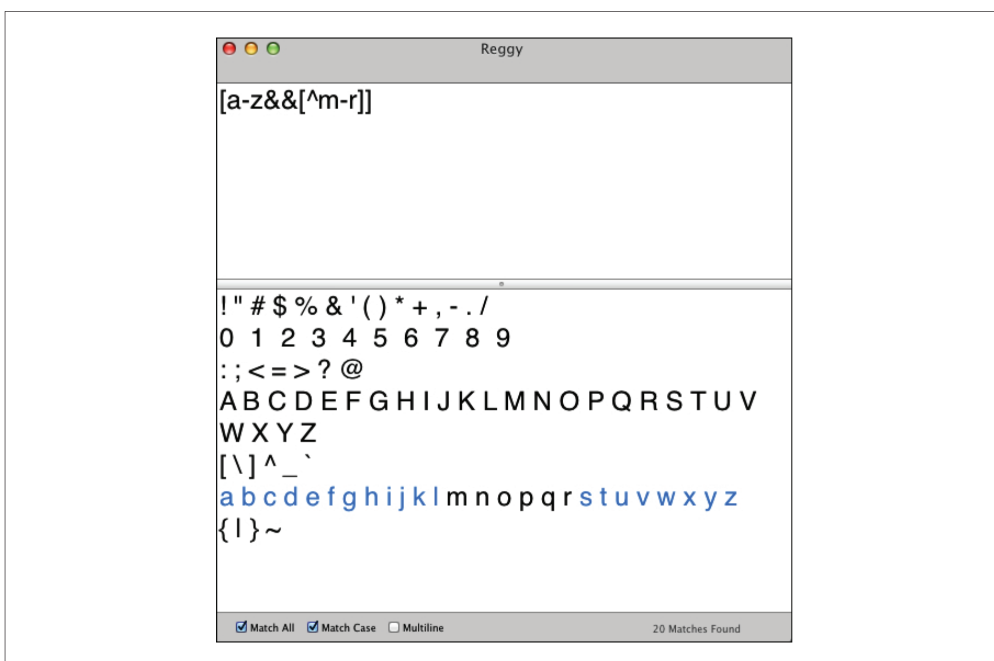


图 5-5：Reggy 中两个字符组的差集

5.3 POSIX字符组

POSIX（Portable Operating System Interface，可移植操作系统接口）是 IEEE 维护的一系列标准。其中包含了一个正则表达式标准（ISO/IEC/IEEE 9945:2009），该标准提供了一套命名的字符组，其形式为：

```
[[:xxxx:]]
```

其中 *xxxx* 是名字，比如 digit 或者 word。

要匹配字母及数字，可以用：

```
[[:alnum:]]
```

图 5-6 展示的是在 Rubular 中使用字母及数字字符组的情况。

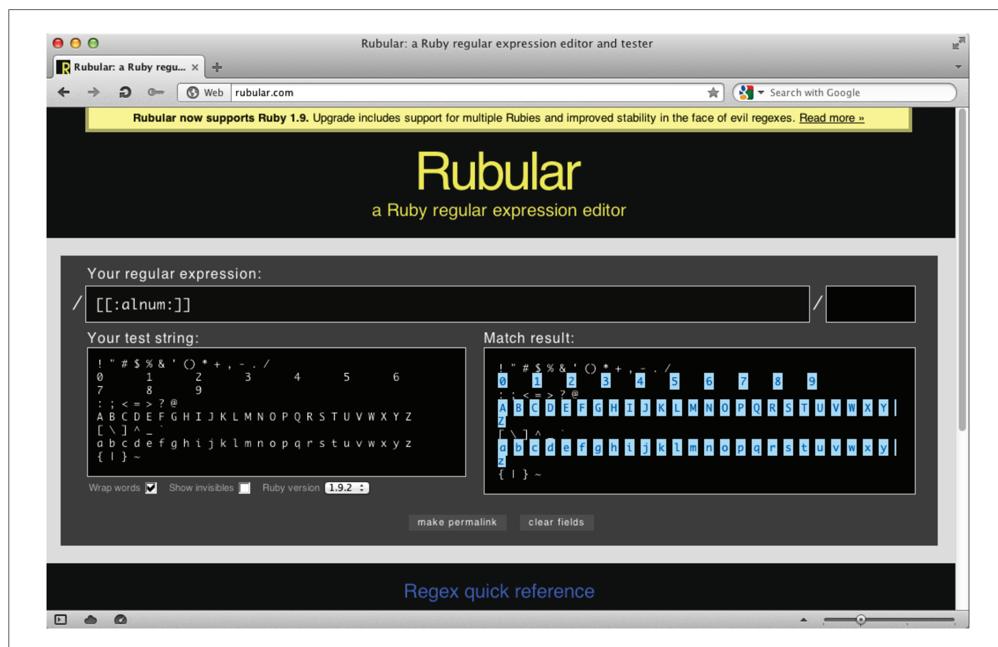


图 5-6：在 Rubular 中使用 POSIX 字母及数字字符组

还有一种方法就是只用简写式 \w。哪种方法写起来简便，是 POSIX 字符组还是简写式？当然是字符最少的方法胜出。我承认自己并不常使用 POSIX 字符组，但是还是有必要了解一下。

对于大写或小写的字母，使用：

```
[[:alpha:]]
```

如果你想匹配 ASCII 范围内的字符，则用：

```
[[:ascii:]]
```

当然，还有对 POSIX 字符组取反的形式：

```
[[:^xxxx:]]
```

如果想匹配非字母字符，可以使用：

```
[[:^alpha:]]
```

要匹配空格或制表符，使用：

```
[[:space:]]
```

若匹配所有空白字符，则用：

```
[[:blank:]]
```

还有很多 POSIX 字符组，如表 5-1 示。

表5-1：POSIX字符组

字符组	描　述
[[:alnum:]]	匹配字母及数字
[[:alpha:]]	匹配字母
[[:ascii:]]	匹配 ASCII 字符（共 128 个）
[[:blank:]]	匹配空白字符
[[:ctrl:]]	匹配控制字符
[[:digit:]]	匹配数字
[[:graph:]]	匹配图形字符
[[:lower:]]	匹配小写字母
[[:print:]]	匹配可打印字符
[[:punct:]]	匹配标点符号
[[:space:]]	匹配空格字符
[[:upper:]]	匹配大写字母
[[:word:]]	匹配单词字符
[[:xdigit:]]	匹配十六进制数字

下一章专门介绍如何匹配 Unicode 和其他字符。

5.4 本章所学

- 如何使用方括号表达式创建字符组或字符集
- 如何在字符组中创建一个或多个范围
- 如何匹配 0 到 99 范围内的偶数
- 如何匹配十六进制数
- 如何在字符组中使用简写式
- 如何对一个字符组进行取反
- 如何得到字符组的并集和差集
- 什么是 POSIX 字符组

5.5 相关资源

- Mac 上的桌面应用 Reggy 可以在 http://www.reggyapp.com 上免费下载。Reggy 通过改变匹配的文本的颜色来显示它所匹配的内容。默认颜色为蓝色，你可以在 Reggy 菜单的 Preference 中修改颜色。具体是在其中的 Regular Expression Syntax（正则表达式语法）中选择 Java。
- Opera Next 浏览器，现在是测试版，可以从 http://www.opera.com/browser/next/ 下载。
- Rubular 是由 Michael Lovitt 创建的在线 Ruby 正则表达式编辑器，它支持 1.8.7 和 1.9.2 版的 Ruby（参见 http://www.rubular.com）。
- 请到 http://mathworld.wolfram.com/EvenNumber.html 阅读更多有关偶数的内容，零也归为偶数。
- http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html 有 Java（1.6）的正则表达式实现文档。
- 可以在 http://www.ieee.org 找到更多有关 IEEE 及其他 POSIX 标准的信息。

第6章

匹配Unicode和其他字符

有时我们需要匹配 ASCII 范围之外的字符。ASCII（American Standard Code for Information Interchange，美国信息交换标准代码）定义了英文字符集（A 到 Z 的大写和小写字母，以及控制字符与其他字符）。它的历史已经很久了，早在 1968 年这个基于拉丁字母的含有 128 个字符的字符集就得到了标准化。那时个人电脑、VisiCalc、鼠标、Web 都还没有出现，而现在我仍然会经常在线查询 ASCII 字符表。

多年以前，我刚开始自己的职业生涯的时候，与一位工程师同事在钱包里放了一张 ASCII 码表。这是以防万一：出门时别忘记带上 ASCII 码表。

我并不否认 ASCII 的重要性，但是现在它已经过时了，尤其是我们有了可以表示超过 10 万个字符的 Unicode 标准（http://www.unicode.org）。然而，Unicode 也没有完全舍弃 ASCII，它将 ASCII 加入了它的基本拉丁（Basic Latin）码表中（参见 http://www.unicode.org/charts/PDF/U0000.pdf）。

本章，我们将跳出 ASCII 的小圈子，投入已经较为普及的 Unicode 的世界。

本章第一个示例文本是代码库中的 voltaire.txt 文件，这是法国哲学家伏尔泰（1694—1778）的一段话。

> Qu’est-ce que la tolérance? c’est l’apanage de l’humanité. Nous sommes tous pétris de faiblesses et d’erreurs; pardonnons-nous réciproquement nos sottises, c’est la première loi de la nature.

翻译成英文就是：

What is tolerance? It is the consequence of humanity. We are all formed of frailty and error; let us pardon reciprocally each other's folly—that is the first law of nature.

这段话的意思是：

什么是宽容？它是人性的产物。我们生来都有缺陷和错误，就让我们原谅彼此的蠢行吧！这才是自然的第一法则。

6.1 匹配Unicode字符

有很多方式可以指定 Unicode 字符（也称为代码点）。为了讲解方便，本书将 Unicode 字符看做 ASCII 字符范围以外的字符，但严格说来这并不准确。

先将伏尔泰的名言输入到 Regexpal（http://www.regexpal.com）中，然后输入这个正则表达式：

```
\u00e9
```

`\u` 之后紧跟十六进制值 00e9（这里不区分大小写，即 00E9 也可以）。00e9 对应十进制值 233，在 ASCII 范围（0~127）之外。

注意在 Regexpal 中字母 é（小写 e 加上一个重音符）被标亮了（参见图 6-1）。这是因为在 Unicode 中 é 就是代码点 U+00E9，所以就被 `\u00e9` 匹配。

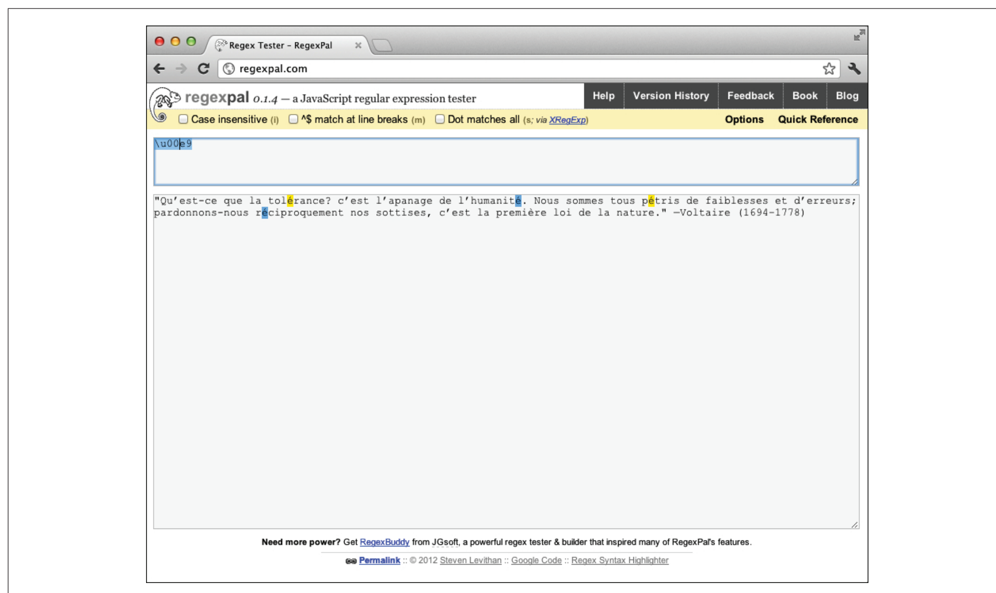


图 6-1：在 Regexpal 中匹配 U+00E9

Regexpal 使用的是 JavaScript 的正则表达式实现。JavaScript 也允许你使用下面的语法：

```
\xe9
```

请在 Regexpal 中试一下，看看它是不是跟 `\u00e9` 一样也匹配同一个字符。

让我们在另一个正则表达式处理引擎中试一下。请在浏览器中打开 http://regexhero.net/tester/。Regex Hero 是用 .NET 编写的且语法稍有不同。将文件 basho.txt 中的内容放入标签为 Target String 的文本区域中。其内容包含日本诗人松尾芭蕉（Matsuo Basho，他恰巧是在伏尔泰出生的前一周去世的）的俳句。

以下是该诗的日文版：

```
古池
蛙飛び込む
水の音
        —芭蕉（1644-1694）
```

接下来是该诗的英译版：

```
At the ancient pond
a frog plunges into
the sound of water.
        —Basho (1644-1694)
```

请在标签为 Regular Expression 的文本区中键入以下内容来匹配日文文本的部分内容：

```
\u6c60
```

这是单词 pond（池塘）所对应的日文字符的代码点，该字会在下方被标亮（参见图 6-2）。

另外，你可以试一下匹配长破折号（—）：

```
\u2014
```

或者短破折号（-）：

```
\u2013
```

现在再在编辑器中看看这些字符。

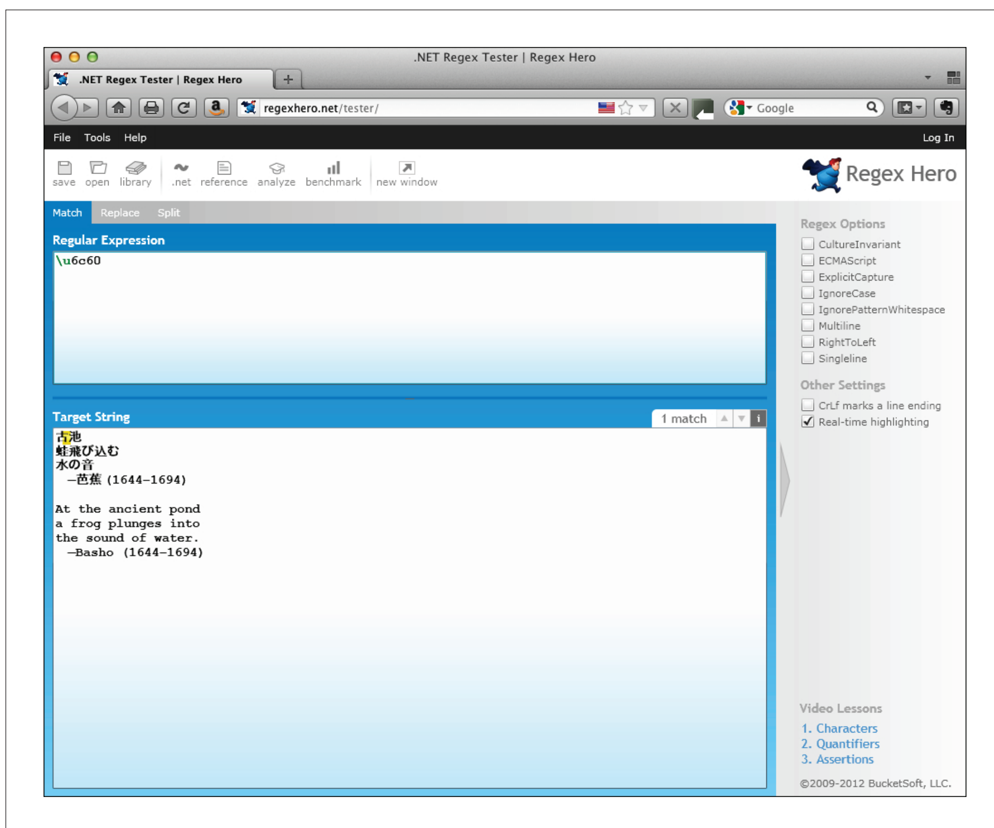


图 6-2：在 Regex Hero 中匹配 U+6c60

使用vim

如果你的系统里有 vim，可以用它打开 basho.txt 文件，如下所示：

```
vim basho.txt
```

现在以斜线（/）为起始，输入下面一行来查找：

```
/\%u6c60
```

然后键入回车（Enter 或 Return）。如图 6-3 所示，光标移到了匹配部分的起始处。表 6-1 列出了可以设置的选项。在 `\%` 之后，你可以使用 `x` 或 `X` 来匹配 0-255（0-FF）范围内的值，使用 `u` 匹配 256–65 535（100–FFFF）范围内的四位十六进制数，还可以用 `U` 来匹配 65 536–2 147 483 647（10000–7FFFFFFF）范围内的八位十六进制数。这样就能涵盖很多编码，其数量远远超过 Unicode 现有的字符编码数量。

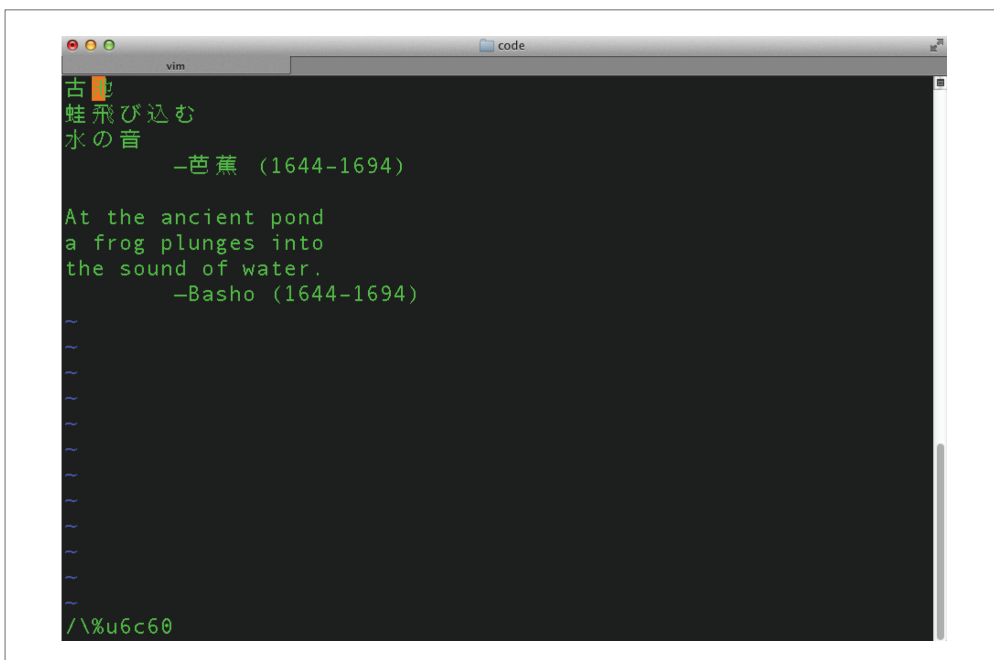


图 6-3：在 Vim 中匹配 U+6c60

表6-1：在Vim中匹配Unicode编码

首　字　符	最大字符数	最　大　值
x 或 X	2	255（FF）
u	4	65 535 (FFFF)
U	8	2 147 483 647 (7FFFFFFF)

6.2　用八进制数匹配字符

还可以使用八进制数来匹配字符，八进制数以 8 为基数，使用数字 0 到 7 计数。在正则表达式处理器中，就是要在反斜线（\）后加三位数字。

比如，以下八进制数：

```
\351
```

等同于：

```
\u00e9
```

请在 Regexpal 中用伏尔泰的文本实验一下。`\351` 匹配 é，键入的内容少了一点儿。

6.3 匹配Unicode字符属性

在某些实现（比如像 Perl）中，还可以匹配 Unicode 的字符属性。这些属性包括字符是否是字母、数字或标点符号。

现在介绍一下 ack，这是一个用 Perl 语言编写的命令行工具，它跟 grep 功能相似（见 http://betterthangrep.com）。系统一般不会附带这个程序，用户需要自己下载并安装。（参见 6.6 节）。

我们要用 ack 来处理席勒 1785 年的作品“An die Freude”的片段（或许你不认识，这是德语）：

```
An die Freude.
Freude, schöner Götterfunken,
Tochter aus Elisium,
Wir betreten feuertrunken
Himmlische, dein Heiligthum.
Deine Zauber binden wieder,
was der Mode Schwerd getheilt;
Bettler werden Fürstenbrüder,
wo dein sanfter Flügel weilt.

Seid umschlungen, Millionen!
Diesen Kuß der ganzen Welt!
Brüder, überm Sternenzelt
muß ein lieber Vater wohnen.
```

该片段中有几个有趣的字符，它们不在 ASCII 的范围内。我们要通过属性来看看这首诗的文本内容。（如果你想知道这段文字的意思，可以将其输入到 Google Translate 中（http://translate.google.com）。）

在命令行中使用 ack 时，你可以指定查看那些属性为字母（L）的字符：

```
ack '\pL' schiller.txt
```

该命令会将字母都高亮显示。对于小写字母，则要在括号中使用 `Ll`：

```
ack '\p{Ll}' schiller.txt
```

括号是必需的。对于大写字母，则是 `Lu`：

```
ack '\p{Lu}' schiller.txt
```

要指定不符合某个属性的字符，则使用大写 `P`：

```
ack '\PL' schiller.txt
```

这个命令会将非字母字符标亮。

下面的正则表达式用于查找非小写字母：

```
ack '\P{Ll}' schiller.txt
```

而这个正则表达式则高亮显示非大写字母：

```
ack '\P{Lu}' schiller.txt
```

在另外一个基于浏览器的正则表达式测试程序 http://regex.larsolavtorvik.com 中也可以这样做。图 6-4 展示的是使用小写字母属性（`\p{Ll}`）标亮席勒诗中小写字母的结果。

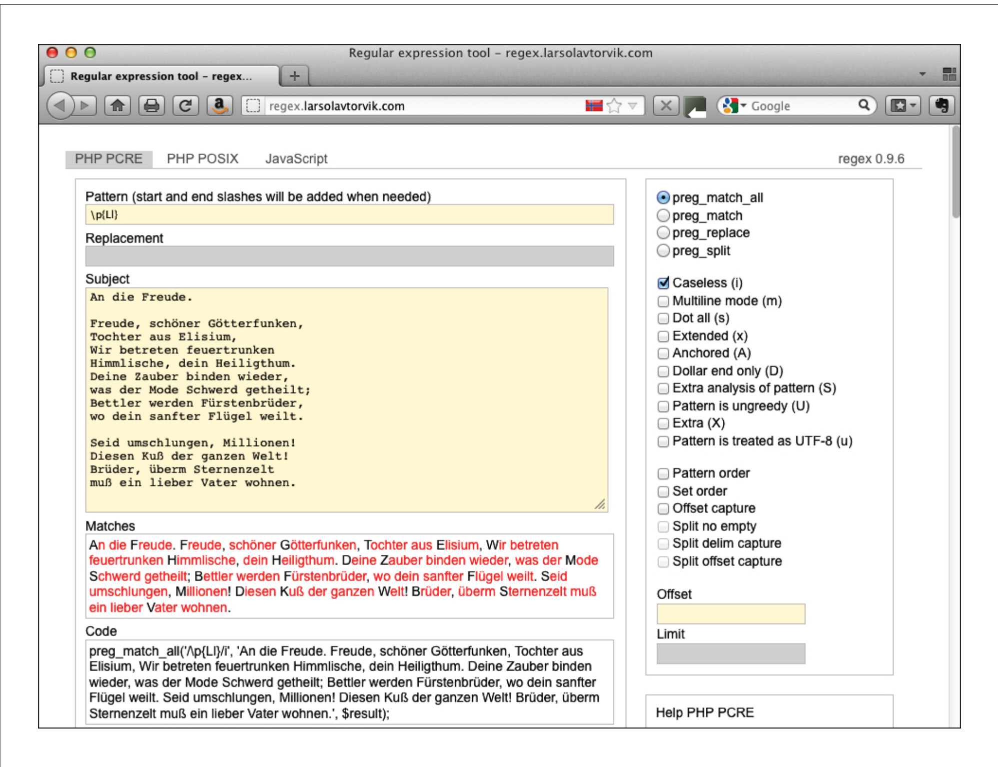


图 6-4：使用小写字母属性标亮字符

表 6-2 列出了在 `\p{property}` 或 `\P{property}` 中使用的字符属性名（参见 http://www.pcre.org/pcre.txt 中的 pcresyntax(3)）。还可以用属性对各种人类语言进行匹配，具体参见表 A-8。

表6-2：字符属性

属　性	描　述	属　性	描　述	属　性	描　述
C	其他字符	M	标记符号	Pi	起始标点符
Cc	控制字符	Mc	空格标记	Po	其他标点符
Cf	格式字符	Me	环绕标记	Ps	开始标点符
Cn	未分配字符	Mn	非空格标记	S	符号
Co	专用字符	N	数字	Sc	货币符号
Cs	替代字符	Nd	十进制数字	Sk	修饰符号
L	字母	Nl	字母数字	Sm	数学符号
Ll	小写字母	No	其他数字	So	其他符号
Lm	修饰字母	P	标点符号	Z	分隔符
Lo	其他字母	Pc	连接标点符	Zl	行分隔符
Lt	标题大写字母	Pd	破折号	Zp	段落分隔符
Lu	大写字母	Pe	结束标点符	Zs	空格分隔符
L&	Ll、Lu 或者 Lt	Pf	最终标点符		

6.4　匹配控制字符

如何匹配控制字符呢？虽然你很少会在文本中查找控制字符，但知道如何做也不是件坏事。在示例代码库中，你会找到 ascii.txt 文件（共 128 行），包含了所有的 ASCII 字符，一个字符占一行（所以有 128 行）。当你在该文件中查找时，若找到匹配项就返回一行。这个文件用来试试手挺不错的。



如果你使用 grep 或 ack 在 ascii.txt 中查找字符串或控制字符，这两个程序可能会将该文件当做二进制文件。如果是这样，在对其执行一个脚本时，若找到匹配程序就会报告“Binary file ascii.txt matches”（二进制文件匹配成功）。就这一点需要注意。

在正则表达式中，可以像这样来指定一个控制字符：

```
\cx
```

其中 x 就是你想匹配的控制字符。

例如，要在一个文件中查找空字符，可以使用以下 Perl 命令：

```
perl -n -e 'print if /\c@/' ascii.txt
```

如果系统中已经安装了 Perl 且运行正常，就可以得到以下结果：

```
0. Null
```

原因是该行有一个空字符，只是结果中看不到这个字符。



如果你用 vim 之外的编辑器打开 ascii.txt 文件，编辑器就有可能将控制字符从文件中删除，所以最好还是使用 vim 编辑器。

还可以用 `\0` 来查找空字符。试一下这个命令：

```
perl -n -e 'print if /\0/' ascii.txt
```

再用以下命令来查找报警字符（BEL）：

```
perl -n -e 'print if /\cG/' ascii.txt
```

这将返回：

```
7. Bell
```

或者还可以使用简写式：

```
perl -n -e 'print if /\a/' ascii.txt
```

要查找转义字符，则使用：

```
perl -n -e 'print if /\c[/' ascii.txt
```

它的结果是：

```
27. Escape
```

或者使用简写式：

```
perl -n -e 'print if /\e/' ascii.txt
```

如何匹配退格符呢？试一下这个：

```
perl -n -e 'print if /\cH/' ascii.txt
```

它显示：

```
8. Backspace
```

也可以使用分类表达式来查找退格符：

```
perl -n -e 'print if /[\b]/' ascii.txt
```

如果没有括号，\b 会被认为是什么呢？是第 2 章中所学到的单词边界。括号改变了正则表达式处理器对 \b 的理解方式。在本例中，Perl 将其看做一个退格符。

表 6-3 列出了本章中匹配字符的方法。

表6-3：匹配Unicode及其他字符

代　码	描　述	代　码	描　述
\uxxxx	Unicode（四位）	\cx	控制字符
\xxx	Unicode（两位）	\0	空字符
\x{xxxx}	Unicode（四位）	\a	报警符
\x{xx}	Unicode（两位）	\e	转义符
\ooo	八进制（基数为 8）	[\b]	退格符

该表总结了本章的内容。下一章我们详细学习量词。

6.5　本章所学

- 如何使用 \uxxxx 或 \xxx 匹配任意 Unicode 字符
- 如何在 vim 中使用 \%xxx、\%Xxx、\%uxxxx 或者 \%Uxxxx 匹配任意 Unicode 字符
- 如何用八进制格式 \ooo 匹配 0-255 范围内的字符
- 如何通过 \p{x} 来使用 Unicode 字符属性
- 如何用 \e 或 \cH 匹配控制字符
- 更多有关如何在命令行中使用 Perl 的内容（更多的单行 Perl 命令）

6.6　相关资源

- 我是使用 vim（http://www.vim.org）将控制字符输入到 ascii.txt 文件中的。在 vim 中，可以在使用 Ctrl+V 之后输入适当的控制字符序列，比如 Ctrl+C 是文本结束字符。我还在 Ctrl+V 后输入字符对应的两位十六进制码。还可以使用二合字母（digraph）来输入控制字符；在 vim 中输入 :digraph 就可看到多个可用编码。要输入二合字母，可在插入（Insert）模式下使用 Ctrl+K 之后输入两个相连的字母（例如，NU 表示空字符）。
- RegexHero（http://regexhero.net/tester）是一个基于 .NET 的正则表达式实现程序，它由 Steve Wortham 编写，在浏览器中运行。这是个收费程序，但可以免费试用，如果你想购买这个程序，它的价格还算合理（有标准版和专业版两种选择）。

- vim（http://www.vim.org）是 Bill Joy 于 1976 年发明的 vi 编辑器的改进版本。它最初由 Bram Moolenaar 开发。对于不熟悉它的人来说，它显得过时了，但就像我提到过的，它功能非常强大。
- 工具程序 ack（http://betterthangrep.com）是使用 Perl 语言编写的。它功能上与 grep 相似且有很多命令行选项，在很多方面都超越了 grep。例如，它使用的是 Perl 正则表达式而不是 grep 之类的基本正则表达式(不使用 -E)。要获取安装指南，请参见 http://betterthangrep.com/install/。我使用的是“Install the ack executable.”下面的安装指南。我没有使用 curl 而只用给出的链接下载了 ack，之后就将该脚本复制到 Mac 和在 Windows 7 中运行 Cygwin（http://www.cygwin.com）的 PC 的 /usr/bin 路径下。

第 7 章

量词

前面介绍过一些量词，而本章来细致地讲解量词。

在下面的示例中，我们将使用第 5 章用过的 Mac 桌面应用程序 Reggy（参见图 7-1）。取消勾选底部的 Match All（匹配所有）选项。

如果你使用的不是 Mac，可以在本书之前提到的某个程序中尝试这些示例。请将 triangle.txt 中排成直角三角形的数字粘贴到程序中。该文件位于示例代码库中。

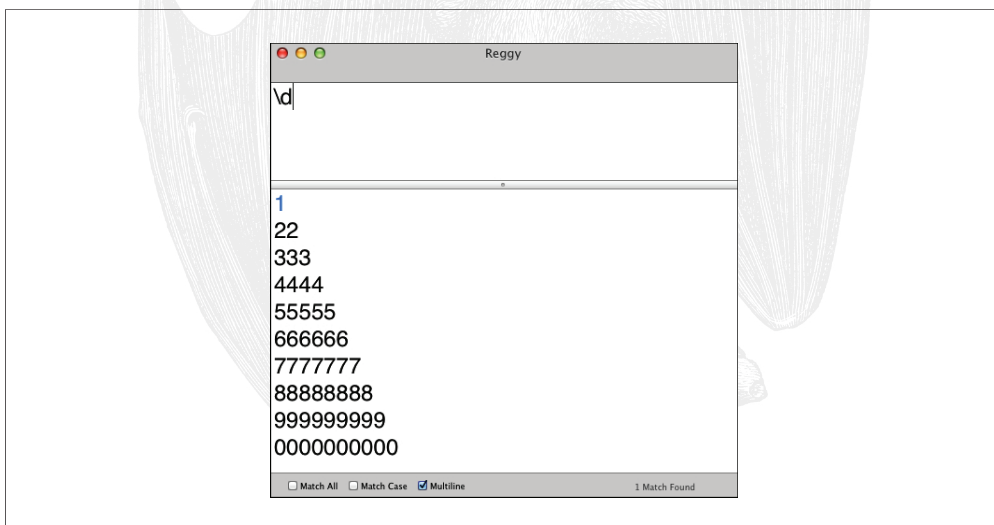


图 7-1：Reggy 应用程序

7.1 贪心、懒惰和占有

这说的可不是小时候的你，而是在说量词。这些形容词听起来不像什么好特性，但要想把正则表达式用好，却一定要了解量词的这些特性。

量词自身是贪心的。贪心的量词会首先匹配整个字符串。尝试匹配时，它会选定尽可能多的内容，也就是整个输入。量词首次尝试匹配整个字符串，如果失败则回退一个字符后再次尝试。这个过程叫做*回溯*（backtracking）。它会每次回退一个字符，直到找到匹配的内容或者没有字符可尝试为止。此外，它还记录所有的行为，因此相较另两种方式它对资源的消耗最大。它先“吃”尽所有的字符，然后每次“吐”出一点，慢慢咀嚼消化……你应该明白了。

懒惰（有时也说*勉强*）的量词则使用另一种策略。它从目标的起始位置开始尝试寻找匹配，每次检查字符串的一个字符，寻找它要匹配的内容。最后，它会尝试匹配整个字符串。要使一个量词成为懒惰的，必须在普通量词后添加一个问号（?）。它每次只“吃”一点。

占有量词会覆盖整个目标然后尝试寻找匹配内容，但它只尝试一次，不会回溯。占有量词就是在普通量词之后添加一个加号（+）。它不“咀嚼”而是直接“吞咽”，然后才想知道“吃”的是什么。下面几节将逐一展示这几种量词。

7.2 用*、+和?进行匹配

如果你在 Reggy 中输入了前面的数字，现在就可以开始测试了。首先使用 Kleene 星号，这一命名是为了纪念正则表达式的发明人 Stephen Kleene。如果像下面这样在点号之后使用星号：

```
.*
```

它将会以贪心的方式匹配主文本中的所有字符（数字）。如前所述，`.*` 匹配任何字符零次或多次。下方文本框中的所有数字都会以另一种颜色高亮显示。一本早期手册对 Kleene 星号的描述如下：

> 正则表达式后跟一个“*”（Kleene星号）表示该正则表达式所匹配的文本接连出现任意次（包括零次）。

现在试一下：

```
9*
```

底部数字 9 的那一行就会被标亮。现在再试一下：

```
9.*
```

这就会将包含数字 9 的行和下面的包含数字 0 的行标亮。因为勾选了 Multiline 选项（在应用程序窗口底部），点号会匹配行之间的换行符；正常情况下，它不会匹配换行符。

要匹配一个或多个 9，尝试：

```
9+
```

跟前面有什么不同呢？暂时看不出来，因为在主文本中有九个 9。主要的区别是 + 会寻找至少一个 9，而 * 会寻找零个或多个 9。

要匹配零次或一次（可选），可使用：

```
9?
```

这只会匹配第一次出现的 9。这个数字 9 是可选的，由于它存在于主文本中，所以它会匹配并被标亮。如果用下面这个表达式：

```
99?
```

则第一个 9 和第二个 9 都会匹配。

表 7-1 列出了基本的量词以及它们的一些可能的作用。这些量词默认是*贪心的*，这意味它们在第一次尝试时会尽可能多地匹配字符。

表7-1：基本量词

语　法	描　述
?	零个或一个（可选）
+	一个或多个
*	零个或多个

7.3　匹配特定次数

使用花括号可以限制某个模式在某个范围内匹配的次数，未经修饰的量词就是贪心量词。例如：

```
7{1}
```

会匹配第一次出现的 7。要匹配一个或多个数字 7，只要加一个逗号即可：

```
7{1,}
```

你可能已经意识到了

```
7+
```

和

```
7{1,}
```

本质上是一样的，而：

```
7*
```

和

```
7{0,}
```

也是相同的。另外，

```
7?
```

与

```
7{0,1}
```

也是一样的。

还可以匹配 *m* 到 *n* 次，比如：

```
7{3,5}
```

会匹配三个、四个以及五个 7。

可以看出，花括号（或者说范围语法）是最灵活和精确的量词。表 7-2 总结了这些特性。

表7-2：范围语法总结

语　法	描　述
`{n}`	精确匹配 *n* 次
`{n,}`	匹配 *n* 次或更多次
`{m,n}`	匹配 *m* 至 *n* 次
`{0,1}`	与 ? 相同 (零次或一次)
`{1,0}`	与 + 相同 (一次或更多)
`{0,}`	与 * 相同 (零次或更多)

7.4 懒惰量词

现在让我们把贪心特性放到一边而来看看懒惰量词。理解懒惰特性最好的方式就是看看实际应用。请在 Reggy（确定 Match All 没有被勾选）中尝试用问号（?）匹配零个或者一个 5：

```
5?
```

第一个 5 被标亮了。请再加一个 ? 来使量词变为懒惰的：

```
5??
```

现在它看起来不匹配任何内容了，其原因是该模式已经是懒惰的了。也就是说，它不会强制匹配第一个 5。懒惰的基本特性就是匹配尽可能少的字符——它就是个“懒虫”。

试一下匹配零次或多次的量词：

```
5*?
```

它也不会匹配任何内容，因为它可以选择匹配最少的次数——零次。

再试一下匹配一次或多次：

```
5+?
```

看到了吧，懒惰特性使其只匹配了一个 5。它只需要做到这个程度就可以了。

使用 *m* 和 *n* 方式匹配时就更为有趣了。请尝试：

```
5{2,5}?
```

只匹配了两个 5，而不像贪心量词那样匹配五个。

表 7-3 列出了懒惰量词。什么时候懒惰式匹配最实用？如果你想匹配最少而不是最多数目的字符，就可以使用懒惰量词。

表7-3：懒惰量词

语　法	描　述
??	懒惰匹配零次或一次（可选）
+?	懒惰匹配一次或多次
*?	懒惰匹配零次或多次
{*n*}?	懒惰匹配 *n* 次
{*n*,}?	懒惰匹配 *n* 次或多次
{*m*,*n*}?	懒惰匹配 *m* 至 *n* 次

7.5 占有量词

占有式匹配很像贪心式匹配，它会选定尽可能多的内容。但与贪心式匹配不同的是它不进行回溯。它不会放弃所找到的内容，它很自私，这也是把它称为占有式（possessive）的原因。它紧紧"抱"住自己所选的内容，一点也不放弃。但占有量词的优点是速度快，因为无需回溯。当然，匹配失败的话也很快。



说实话，用本书中的例子你很难看出贪心式、懒惰式以及占有式量词的区别。但随着经验的增长，以及对性能的看重，你会发觉其中的不同。

为了理解这一点，我们先尝试匹配以零开头的多个零，然后再匹配以零结尾的多个零。在 Reggy 中，先确定勾选 Match All，然后输入以下以零开头的表达式：

```
0.*+
```

发生什么了？所有的零都被标亮了。存在一个匹配。占有式的匹配看起来和贪心式的匹配是一样的，但没有回溯。可以证明一下。输入这个带有结尾零的表达式：

```
.*+0
```

没有匹配——原因就是没有回溯。它一下就选定了所有的输入，不再回过来查看。它"挥霍"了自己的财产。它一下子没在结尾找到零，也不知道该从哪里找起。如果将加号去掉，它会找到所有的 0，因为它变回贪心式匹配了。

```
.*0
```

当你知道文本中的内容时，就知道在哪里可以找到匹配，这时你应该会使用占有量词。你不在乎它是否会选定所有内容。占有式匹配有助于提高匹配的性能。表 7-4 列出了占有量词。

表7-4：占有量词

语　法	描　述
`?+`	占有式匹配零次或一次（可选）
`++`	占有式匹配一次或多次
`*+`	占有式匹配零次或多次
`{n}+`	占有式匹配 *n* 次
`{n,}+`	占有式匹配 *n* 次或更多次
`{m,n}+`	占有式匹配 *m* 至 *n* 次

下一章将会介绍环视。

7.6　本章所学

- 贪心式、懒惰式以及占有式匹配的区别
- 如何匹配一次或多次（+）
- 如何进行可选匹配（?，零次或一次）
- 如何匹配零次或多次（*）
- 如何使用 {*m*, *n*} 量词
- 如何使用贪心、懒惰（勉强）以及占有量词

7.7　相关资源

引文出自 Dennis Ritchie 和 Ken Thompson，*QED Text Editor*（1970 年贝尔实验室出版）p.3（参见 http://cm.bell-labs.com/cm/cs/who/dmr/qedman.pdf）。

第 8 章

环视

环视是一种非捕获分组，它根据某个模式之前或之后的内容匹配其他模式。环视也称为零宽度断言。

环视包括：

- 正前瞻；
- 反前瞻；
- 正后顾；
- 反后顾。

本章将逐个展示这几种环视的用法。开始先用桌面版的 RegExr，然后再使用 Perl 和 ack（grep 没有环视的概念）。目标文本仍然使用柯勒律治的诗。

8.1 正前瞻

假设要匹配单词 ancyent，且要求紧随其后的单词是 marinere（为与文件中的单词一致，我采用了古代拼写法）。要达到这个目的，我们可以使用正前瞻。

首先在 RegExr 桌面版程序中试一下。将下面这个不区分大小写的模式输入到顶部的文本框内：

```
(?i)ancyent (?=marinere)
```



在 RegExr 中，还可以勾选 ignoreCase 旁的复选框来指定不区分大小写，这两种方式都是可以的。

由于我们使用了不区分大小写的选项（`?i`），就不必担心模式中用的大小写形式了。现在就是在每一行中寻找后跟 marinere 的单词 ancyent。结果会在模式区域下面的文本区中标亮（参见图 8-1）；但只有模式的第一部分（ancyent）是被标亮的，环视模式（Marinere）不会标亮。

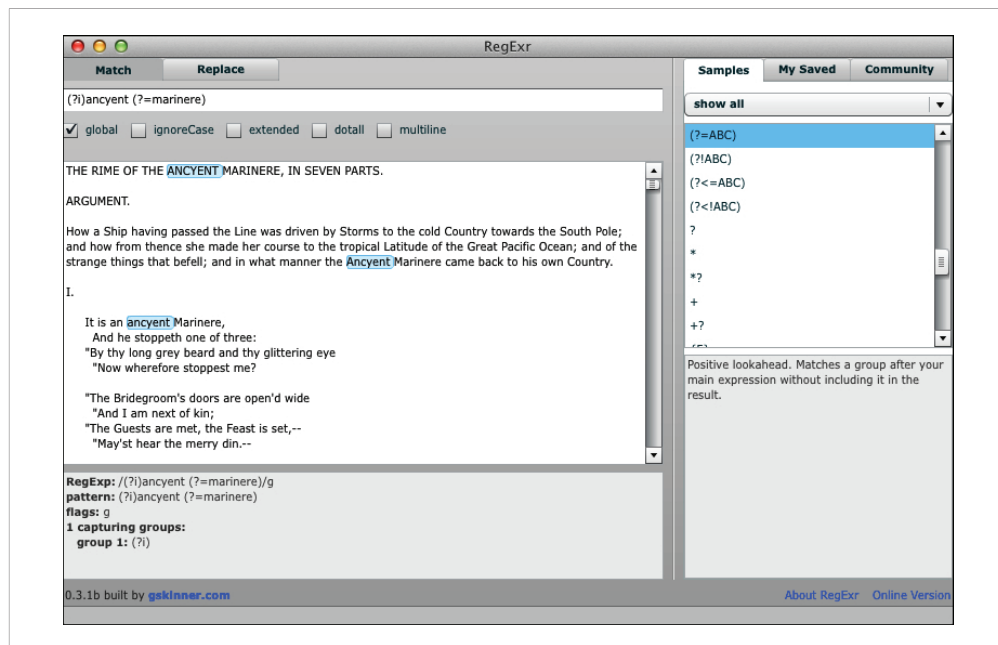


图 8-1：RegExr 中的正前瞻

现在使用 Perl 来做正前瞻。你可以写这样一个命令：

```
perl -ne 'print if /(?i)ancyent (?=marinere)/' rime.txt
```

其输出如下：

```
THE RIME OF THE ANCYENT MARINERE, IN SEVEN PARTS.
How a Ship having passed the Line was driven by Storms to the cold Country
towards the South Pole; and how from thence she made her course to the tropical
Latitude of the Great Pacific Ocean; and of the strange things that befell;
and in what manner the Ancyent Marinere came back to his own Country.
     It is an ancyent Marinere,
     "God save thee, ancyent Marinere!
     "I fear thee, ancyent Marinere!
```

该诗有五行内容中的 ancyent 出现在 marinere 之前。如果要求 ancyent 之后的单词以大写或小写字母 m 开头，该如何做呢？可以这样：

```
perl -ne 'print if /(?i)ancyent (?=m)/' rime.txt
```

现在除了 Marinere，跟在后面的还可以有 man 和 Man：

```
And thus spake on that ancyent man,
And thus spake on that ancyent Man,
```

ack 也可以使用环视功能，这是因为它是由 Perl 语言编写的。ack 的命令行界面与 grep 十分相似。

试一下这个命令：

```
ack '(?i)ancyent (?=ma)' rime.txt
```

你会看到高亮显示的结果，如图 8-2 所示。

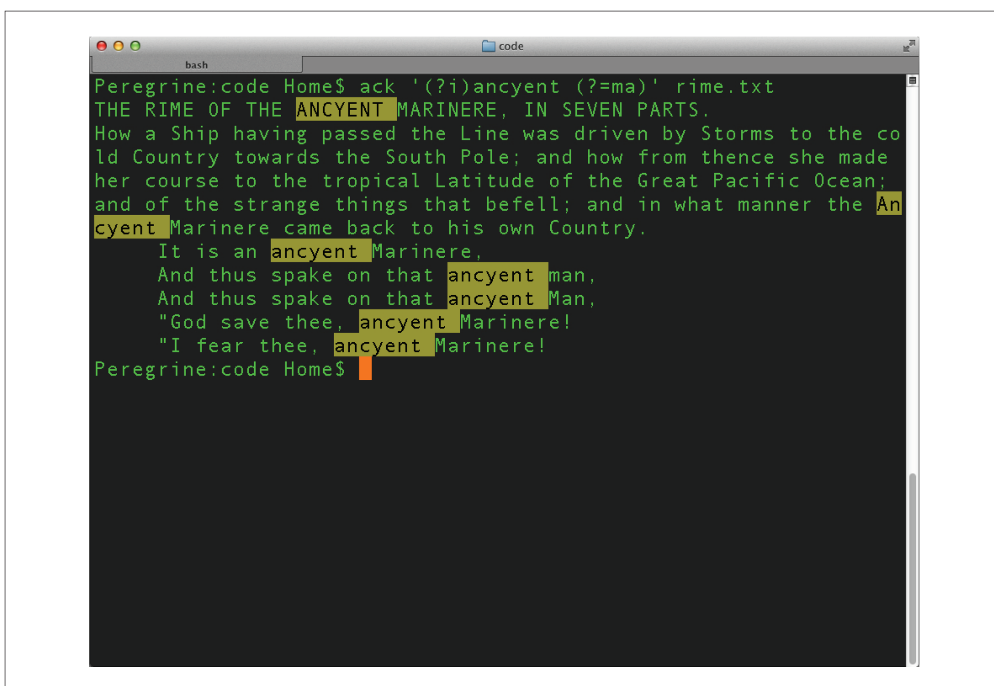


图 8-2：在终端中使用 ack 进行正前瞻查找

使用 ack 时，可以用命令行选项 `-i` 指定不区分大小写，而不使用嵌入选项 `(?i)`：

```
ack -i 'ancyent (?=ma)' rime.txt
```

若要在 ack 的输出中添加行号以方便计数，可以采用多种方法。比如可以加上 -H 选项：

```
ack -Hi 'ancyent (?=ma)' rime.txt
```

也可以添加带有 --output 选项的代码：

```
ack -i --output '$.:$_' 'ancyent (?=ma)' rime.txt
```

这有点不太正统——关闭了标亮功能，但它确实起作用了。

8.2 反前瞻

反前瞻是对正前瞻的取反操作。这意味着要匹配某个模式时，需要在它后面找不到含有给定前瞻模式的内容。反前瞻的形式是：

```
(?i)ancyent (?!marinere)
```

只有一个字符发生了变化：正前瞻的等号（=）变为反前瞻的感叹号（!）。图 8-3 所示为在 Opera 中执行反前瞻查找。

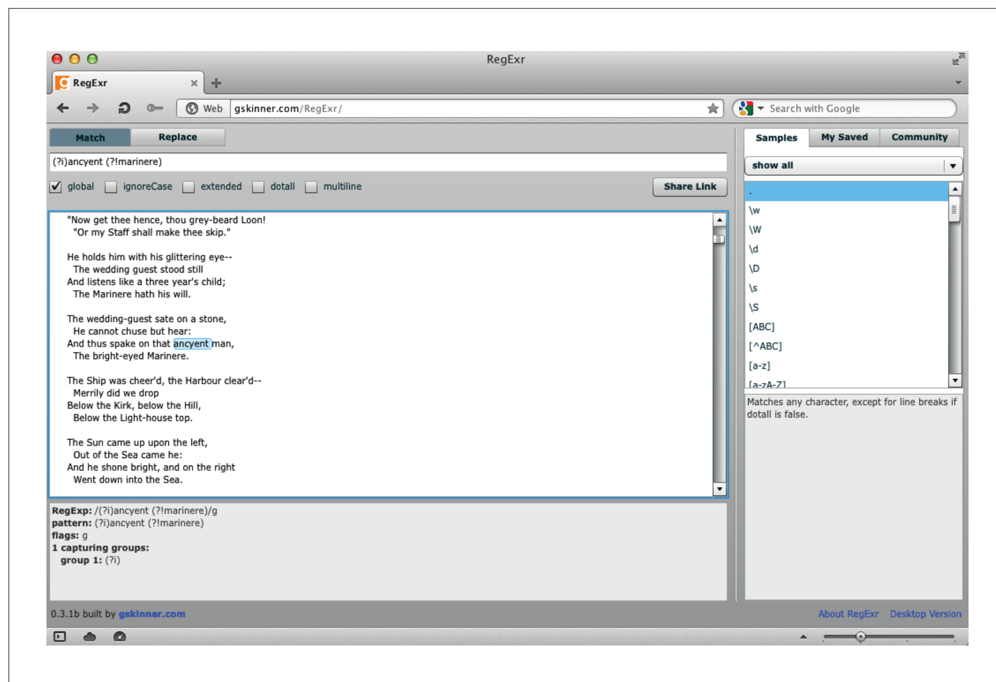


图 8-3：Opera 中使用 RegExr 的反前瞻

在 Perl 语言中，可以这样来使用反前瞻：

```
perl -ne 'print if /(?i)ancyent (?!marinere)/' rime.txt
```

而我们会得到的结果是：

```
And thus spake on that ancyent man,
And thus spake on that ancyent Man,
```

在 ack 中，可以用

```
ack -i 'ancyent (?!marinere)' rime.txt
```

得到相同结果。

8.3 正后顾

正后顾会查看左边的内容，这与正前瞻方向相反。其语法是：

```
(?i)(?<=ancyent) marinere
```

正后顾使用小于号（<），提醒你后顾是哪个方向。请在 RegExr 中试一下，看看有什么不同。被标亮的是 marinere 而不是 ancyent。为什么？因为正后顾的模式是匹配的条件，不会包含在匹配结果中。

在 Perl 语言中这样做：

```
perl -ne 'print if /(?i)(?<=ancyent) marinere/' rime.txt
```

在 ack 中这样做：

```
ack -i '(?<=ancyent) marinere' rime.txt
```

8.4 反后顾

最后一种环视是反后顾。你能猜到它的工作原理吗？

反后顾会查看某个模式在从左至右的文本流的后面没有出现。同样，它有一个小于号（<），提醒后顾的是哪个方向。

请在 RegExr 中使用这个命令并查看结果：

```
(?1)(?<!ancyent) marinere
```

滚动到下方看看结果是什么。

然后用 Perl 语言尝试：

```
perl -ne 'print if /(?i)(?<!ancyent) marinere/' rime.txt
```

以下就是你会看到的结果，没有一个地方含有单词 ancyent：

```
    The Marinere hath his will.
    The bright-eyed Marinere.
    The bright-eyed Marinere.
  The Marineres gave it biscuit-worms,
    Came to the Marinere's hollo!
    Came to the Marinere's hollo!
  The Marineres all 'gan work the ropes,
  The Marineres all return'd to work
  The Marineres all 'gan pull the ropes,
    "When the Marinere's trance is abated."
  He loves to talk with Marineres
  The Marinere, whose eye is bright,
```

最后在 ack 中这样做：

```
ack -i '(?<!ancyent) marinere' rime.txt
```

以上就是对前瞻和后顾的简单介绍，这是现代正则表达式的一个重要特性。

在下一章中，你将会看到如何使用 sed 和 Perl 为文档添加 HTML5 标签的完整例子。

8.5　本章所学

- 如何使用正前瞻和反前瞻
- 如何使用正后顾和反后顾

8.6　相关资源

参见 *Mastering Regular Expressions, Third Edition*[1] 的 59~66 页。

注 1：本书中文版《精通正则表达式（第 3 版）》已经由电子工业出版社出版。——编者注

第 9 章

用HTML标记文档

本章会逐步指导你通过正则表达式为普通文本添加 HTML5 标签，同时总结本书前面所学内容。

如果是我，我就会使用为文本添加 AsciiDoc[1] 标签。但为了我们的示例，还是假设 AsciiDoc 不存在吧（真是太遗憾了！）。我们要用手头仅有的几个工具辛勤地工作——也就是 sed 和 Perl，还有我们自己的智慧。

我们仍然使用 rime.txt 中柯勒律治的诗文作为示例文本。



由于我们对 rime.txt 的结构已经非常了解了，所以本章的脚本是专门针对它写的。如果用这些脚本匹配其他文本，最终的结果可能不同，但这些脚本是你掌控复杂文本结构的良好开端。

9.1　匹配标签

在开始对诗文进行匹配之前，先讨论一下如何匹配 HTML 或 XML 标签。有很多方式匹配标签，无论是起始标签（如 `<html>`）还是结束标签（如 `</html>`），但我找到一个比较可靠的方法。不论是否有属性，它都会匹配起始标签：

注 1：AsciiDoc 是一种轻量级的标记语言，由 Stuart Rackham 最初发布于 2002 年。AsciiDoc 可用于编写文档、电子书、幻灯片、网页、博客等。而且，该格式可以转换成 HTML、PDF、ePub 等格式。要了解有关 AsciiDoc 的更多信息，请参考：http://asciidoc.org/。要了解标记语言的更多信息，请参考：http://en.wikipedia.org/wiki/Comparison_of_documentation_generators。——编者注

```
<[_a-zA-Z][^>]*>
```

以下是该表达式的解析。

- 第一个字符是左尖括号（<）。
- 在 XML 中元素可以以下划线字符（_）开头，而在 HTML 中是以 ASCII 范围中的大写或小写字母开头（参见 9.6 节）。
- 在起始字符之后，标签名称可以是零或多个除右尖括号（>）之外的任意字符。
- 该表达式以右尖括号结尾。

请用 grep 尝试下面的命令。对代码库中的示例 DITA 文件 lorem.dita 进行匹配：

```
grep -Eo '<[_a-zA-Z][^>]*>' lorem.dita
```

结果是：

```
<topic id="lorem">
<title>
<body>
<p>
<p>
<ul>
<li>
<li>
<li>
<p>
<p>
```

要同时匹配起始标签和结束标签，只需添加一个斜线并在之后加一个问号即可。问号使斜线成为可选匹配部分：

```
</?[_a-zA-Z][^>]*>
```

我只在这里讨论起始标签。为了让输入更美观，我常使用管道将结果传入其他工具程序：

```
grep -Eo '<[_a-zA-Z][^>]*>' lorem.dita | sort | uniq | sed 's/^<//;s/ id=
    \".*\"//;s/>$//'
```

该命令列出了排序后的一串 XML 标签名：

```
body
li
p
p
title
topic
ul
```

第 10 章也就是最后一章会更详细地讲解这些内容。接下来几节让我们温故知新。

9.2　用sed转换普通文本

下面我们就为 rime.txt 中的文本添加一些标签，为此可以使用插入命令（`i\`）。找到 rime.txt 文件所在的路径，在 shell 提示符中输入下面的命令：

```
sed '1 i\
<!DOCTYPE html>\
<html lang="en">\
<head>\
<title>The Rime of the Ancyent Marinere (1798)</title>\
<meta charset="utf-8"/>\
</head>\
<body>\

q' rime.txt
```

按回车（或 Return 键），输出如下所示，可以看到加在上面的标签：

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>The Rime of the Ancyent Marinere (1798)</title>
<meta charset="utf-8"/>
</head>
<body>
THE RIME OF THE ANCYENT MARINERE, IN SEVEN PARTS.
```

你刚刚所键入的命令并不会实际改变文件——它只会在屏幕上产生输出。后面我会展示如何修改文件内容。

9.2.1　用sed进行替换

在下面的示例中，我们要用 sed 找出文件的第一行然后使用转义括号 `\(` 和 `\)` 构成的捕获分组捕获一整行。sed 需要对括号进行转义后才可捕获分组，除非你使用 `-E` 选项（稍后讨论）。行起始部分由 `^` 标示，行结束部分由 `$` 标示。后向引用 `\1` 将捕获的文本放入 `title` 元素中，并缩进一个空格。

运行以下命令：

```
sed '1s/^\(.*\)$/ <title>\1<\/title>/;q' rime.txt
```

输出结果如下所示：

```
<title>THE RIME OF THE ANCYENT MARINERE, IN SEVEN PARTS.</title>
```

现在试一下这样：

```
sed -E '1s/^(.*)$/<!DOCTYPE html>\
<html lang="en">\
<head>\
 <title>\1<\/title>\
<\/head>\
<body>\
<h1>\1<\/h1>\
/;q' rime.txt
```

以下是详细分析。

- `-E` 选项代表 sed 使用扩展的正则表达式(也就是 ERE,因此不必对括号进行转义)。
- 使用替换命令时,将第 1 行放入捕获分组(`^(.*)$`)因此可以通过 `\1` 重用该内容。
- 创建 HTML 标签以及用 \ 对换行符进行转义。
- 在 `title` 和 `h1` 之间用 `\1` 插入捕获的文本。
- 在 `q` 处结束程序来防止在屏幕上打印剩下的诗文。

正确的结果是:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>THE RIME OF THE ANCYENT MARINERE, IN SEVEN PARTS.</title>
</head>
<body>
<h1>THE RIME OF THE ANCYENT MARINERE, IN SEVEN PARTS.</h1>
```

9.2.2 用sed处理罗马数字

诗文分为七个部分，每一部分以一个罗马数字开头。还有一个“ARGUMENT”标题。下面一行命令会使用 sed 捕获标题和罗马数字，并将它们用 `<h2>` 标签包括：

```
sed -En 's/^(ARGUMENT\.|I{0,3}V?I{0,2}\.)$/<h2>\1<\/h2>/p' rime.txt
```

以下就是你会看到的结果：

```
<h2>ARGUMENT\.</h2>
<h2>I.</h2
<h2>II.</h2
<h2>III.</h2
<h2>IV.</h2
<h2>V.</h2
<h2>VI.</h2
<h2>VII.</h2
```

接下来是对以上 sed 命令的描述。

- `-E` 选项会使用扩展的正则表达式，而 `-n` 选项会覆盖 sed 默认打印每一行的行为。

- 替换命令（s）会捕获标题和七个大写罗马数字，其中每一个单独一行紧跟一个句号，范围为 I 到 VII。
- s 命令随后将每一行捕获的文本嵌入 h2 元素中。
- 替换部分末尾的 p 标志将结果打印到屏幕上。

9.2.3 用sed处理特定段落

接下来这行命令会找到第 5 行的段落：

```
sed -En '5s/^([A-Z].*)$/<p>\1<\/p>/p' rime.txt
```

然后将该段落放入 <p> 标签中：

```
<p>How a Ship having passed the Line was driven by Storms to the cold Country
      towards the South Pole; and how from thence she made her course to
      the tropical Latitude of the Great Pacific Ocean; and of the strange
      things that befell; and in what manner the Ancyent Marinere came back
      to his own Country.</p>
```

进展有点儿慢？别急，我们很快就会把这些技巧综合起来。

9.2.4 用sed处理多行诗文

接着我们使用下面的表达式来标记多行诗文：

```
sed -E '9s/^[ ]*(.*)/ <p>\1<br\/>/;10,832s/^([ ]{5,7}.*)/\1<br\/>/;
       833s/^(.*)/\1<\/p>/' rime.txt
```

这些 sed 命令是根据行号来实现操作的。通常情况下这种方法并不适用，但若所处理的内容已知则这种方法还是很好的。

- 第 9 行（诗文的第一行，s 命令会选定该行），在文字前面添加几个空格，再插入一个 <p> 标签，然后在行尾添加一个
 标签。
- 第 10 行到第 832 行，每个开头有 5 至 7 个空格的行之后都添加一个
 标签。
- 在第 833 行（诗文的最后一行），s 命令添加 </p> 标签而不是
 标签。

这里是标记后的部分结果：

```
<p>It is an ancyent Marinere,<br/>
     And he stoppeth one of three:<br/>
  "By thy long grey beard and thy glittering eye<br/>
     "Now wherefore stoppest me?<br/>

"The Bridegroom's doors are open'd wide<br/>
  "And I am next of kin;<br/>
"The Guests are met, the Feast is set,--<br/>
  "May'st hear the merry din.--<br/>
```

还应该用
 标签替代空行来分隔诗文：

```
sed -E 's/^$/<br\/>/' rime.txt
```

下面是操作的结果：

```
He prayeth best who loveth best,
  All things both great and small:
For the dear God, who loveth us,
  He made and loveth all.
<br/>
     The Marinere, whose eye is bright,
       Whose beard with age is hoar,
     Is gone; and now the wedding-guest
       Turn'd from the bridegroom's door.
<br/>
     He went, like one that hath been stunn'd
       And is of sense forlorn:
     A sadder and a wiser man
       He rose the morrow morn.
```

我发现在合适的位置添加标签和空格实在太常见了。希望你也能做好。

9.3 追加标签

现在我们要在诗文的结尾添加一些标签。使用追加命令（a\），$ 会查找文件的结尾（最后一行），然后在最后一行添加 </body> 和 </html> 标签：

```
sed '$ a\
<\/body>\
<\/html>\
' rime.txt
```

以下就是文件结尾部分：

```
     He went, like one that hath been stunn'd
       And is of sense forlorn:
     A sadder and a wiser man
       He rose the morrow morn.
</body>
</html>
```

关于 sed 的内容就讲到这里了。

要是想同时完成这些修改呢？你知道怎么做了，而且也做完了。其实只要将这些命令放入一个文件中，然后使用 sed 的 -f 选项即可。

使用sed命令文件

以下是 html.sed 文件的内容，包含前面所有的 sed 命令和另外两个命令。我们将用这个命令文件通过 sed 把 rime.txt 转换为 HTML。文件中的编号便于解释这个 sed 脚本。

```
#!/usr/bin/sed ❶

1s/^(.*)$/<!DOCTYPE html>\ ❷
<html lang="en">\
<head>\
 <title>\1<\/title>\
<\/head>\
<body>\
<h1>\1<\/h1>\
/

s/^(ARGUMENT|I{0,3}V?I{0,2})\.$/<h2>\1<\/h2>/ ❸
5s/^([A-Z].*)$/<p>\1<\/p>/ ❹
9s/^[ ]*(.*)/ <p>\1<br\/>/ ❺
10,832s/^([ ]{5,7}.*)/\1<br\/>/ ❻
833s/^(.*)/\1<\/p>/ ❼
13,$s/^$/<br\/>/ ❽
$ a\ ❾
<\/body>\
<\/html>\
```

❶ 命令的首行是 shebang 行，它提示 shell 可执行文件（sed）的位置。

❷ 替换命令（s）会用随后的标签替换该行文本。反斜线（\）表示你要添加的文本会延续到下一行，因此要插入换行符。用 `\1` 将第 1 行诗文的题目作为 `title` 和 `h1` 的内容插入。

❸ 用 `h2` 标签包括标题和罗马数字。

❹ 第 5 行中用 `p` 标签包括介绍段落。

❺ 第 9 行中在起始处添加 `p` 起始标签，在行尾添加 `br` 标签。

❻ 第 9 行至第 832 行，在以多个空格起始的行的结尾处添加 `br` 标签。

❼ 在诗的结尾附加 `p` 结束标签。

❽ 在第 13 行后，每一个空行都用 `br` 标签代替。

❾ 在文件结尾（$）附加几个标签。

要对 rime.txt 使用该命令文件，则输入下面一行，然后回车（Enter 或 Return）：

```
sed -E -f html.sed rime.txt
```

将输出重定位到文件：

```
sed -E -f html.sed rime.txt > rime.html
```

在浏览器中打开 rime.html 看看你创建的内容（见图 9-1）。

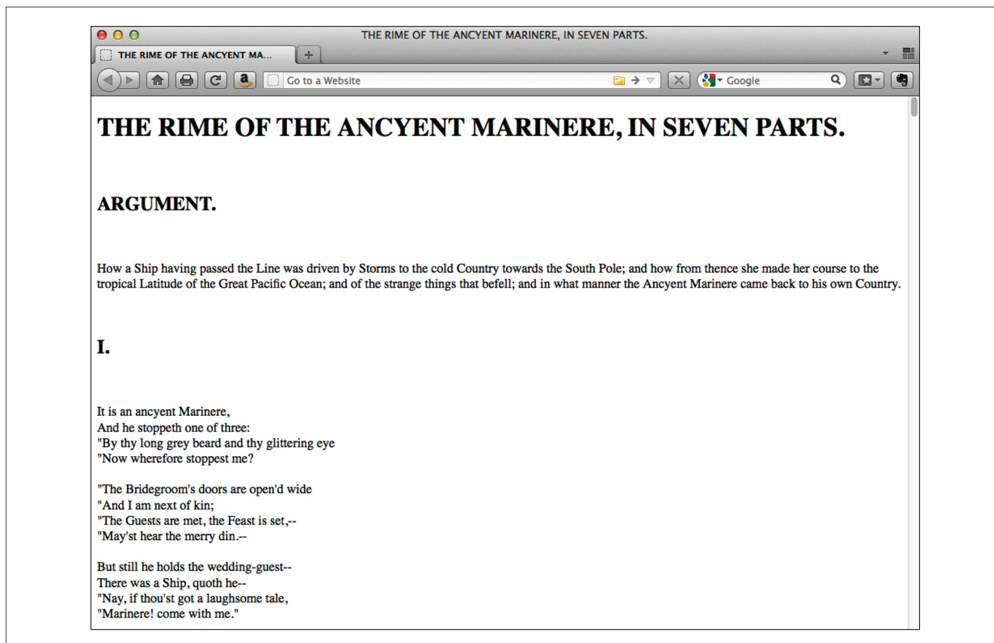


图 9-1：在 Firefox 中打开 rime.html

9.4 用Perl转换普通文本

再看看如何通过 Perl 语言为文本添加 HTML 标签。首先，和 sed 相似，我先逐一给出一系列命令；然后再在一个文件中展示同样的命令。



本书只介绍了 Perl 语言的初步知识及用法。本书不是 Perl 语言教程或手册，但我希望它能激发你对 Perl 语言的兴趣，因此想多展示几种可能的用法。学习 Perl 语言最好的起点是 Learning Perl 网站 http://learn.perl.org/，上面还有 Perl 的安装指导方法。

如果当前行（`$.`）就是第 1 行，将整行（`$_`）赋值给变量 `$title`，再将 `$title` 打印出来。

```
perl -ne 'if ($. == 1) {chomp($title = $_); print "<h1>" . $title . "</h1>" .
    "\n";};'rime.txt
```

如果一切正常，结果应该是：

```
<h1>THE RIME OF THE ANCYENT MARINERE, IN SEVEN PARTS.</h1>
```

以下是对这个 Perl 命令的详解。

- 通过 `$.` 检查是否在第 1 行。
- 选定整行（`$_`）然后将这个字符串赋值给 `$title` 变量。当用 `chomp` 函数选定整行时，就会将结尾的换行符从字符串中剔除。
- 在 `h1` 元素中打印 `$title`，然后再打印换行符（`\n`）。



要了解关于 Perl 语言内置变量（如 `$.`）的更多信息，请在提示符中输入命令 `perldoc -v $.`（perldoc 通常在安装 Perl 时就已经安装了）。如果该命令无效，请参见 9.6 节。

要在文件起始处添加标记（包括 `h1` 标签），使用以下命令：

```
perl -ne 'if ($. == 1) {chomp($title = $_)};
print "<!DOCTYPE html>\
<html xmlns=\"http://www.w3.org/1999/xhtml\">\
 <head>\
  <title>$title</title>\
  <meta charset=\"utf-8\"/>\
 </head>\
<body>\
<h1>$title</h1>\n" if $. == 1; exit' rime.txt
```

结果如下：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <title>THE RIME OF THE ANCYENT MARINERE, IN SEVEN PARTS.</title>
  <meta charset="utf-8"/>
 </head>
<body>
<h1>THE RIME OF THE ANCYENT MARINERE, IN SEVEN PARTS.</h1>
```

打印函数 `print` 会打印随后的标签以及每一行，并在行尾加 \，其中 \ 会将换行符写入到输出内容中。`$title` 变量的内容会在 `title` 和 `h1` 元素中展开显示。

9.4.1 用Perl处理罗马数字

要给标题和用于分隔内容的罗马数字添加标签，可使用：

```
perl -ne 'print if s/^(ARGUMENT\.|I{0,3}V?I{0,2}\.)$/<h2>\1<\/h2>/;' rime.txt
```

输出如下所示：

```
<h2>ARGUMENT.</h2>
<h2>I.</h2>
<h2>II.</h2>
<h2>III.</h2>
<h2>IV.</h2>
<h2>V.</h2>
<h2>VI.</h2>
<h2>VII.</h2>
```

替换命令（s）会捕获 ARGUMENT 标题和七个大写罗马数字，其中每一个都在单独的一行且尾随一个句点，罗马数字范围为 I 至 VII；然后将捕获的文本包括在 h2 标签中。

9.4.2 用Perl处理特定段落

请使用以下代码在行号为 5 时用 p 元素包括介绍段落：

```
perl -ne 'if ($. == 5) {s/^([A-Z].*)$/<p>$1<\/p>/;print;}' rime.txt
```

可以看到如下内容：

```
<p> How a Ship having passed the Line was driven by Storms to the cold Country
       towards the South Pole; and how from thence she made her course to
       the tropical Latitude of the Great Pacific Ocean; and of the strange
       things that befell; and in what manner the Ancyent Marinere came back
       to his own Country.</p>
```

9.4.3 用Perl处理多行诗文

以下命令会将一个 p 起始标签放在诗文第一行的起始处，并在该行的结尾添加一个 br 标签：

```
perl -ne 'if ($. == 9) {s/^[ ]*(.*)/ <p>$1<br\/>/;print;}' rime.txt
```

结果如下：

```
<p>It is an ancyent Marinere,<br/>
```

接下来的 Perl 命令会在第 10 行到第 832 行之间的每一行诗文的结尾处添加一个 br 标签：

```
perl -ne 'if (10..832) { s/^([ ]{5,7}.*)/$1<br\/>/; print;}' rime.txt
```

下面是你可以看到的结果的一部分：

```
Farewell, farewell! but this I tell<br/>
  To thee, thou wedding-guest!<br/>
He prayeth well who loveth well<br/>
  Both man and bird and beast.<br/>
```

在诗文最后一行结尾处添加一个 p 结束标签。

```
perl -ne 'if ($. == 833) {s/^(.*)/$1<\/p>/; print;}' rime.txt
```

这就会显示：

```
He rose the morrow morn.</p>
```

将每一行结尾处的空行替换为 br 标签：

```
perl -ne 'if (9..eof) {s/^$/<br\/>/; print;}' rime.txt
```

结果为：

```
<br/>
    He prayeth best who loveth best,
      All things both great and small:
    For the dear God, who loveth us,
      He made and loveth all.
<br/>
    The Marinere, whose eye is bright,
      Whose beard with age is hoar,
    Is gone; and now the wedding-guest
      Turn'd from the bridegroom's door.
<br/>
```

最后，找到文件结尾时，就打印出几个结束标签：

```
perl -ne 'if (eof) {print "</body>\n</html>\n"};' rime.txt
```

将所有这些代码放在一个文件中运行起来更简单，下一节就会看到了。

9.4.4 使用Perl命令文件

下面列出的是 html.pl 文件，它用 Perl 语言将 rime.txt 转换为 HTML。该示例中的编号将指导你理解这个 Perl 脚本的内容。

```
#!/usr/bin/perl -p ❶

if ($. == 1) { ❷
 chomp($title = $_);
}
print "<!DOCTYPE html>\ ❸
<html xmlns=\"http://www.w3.org/1999/xhtml\">\
 <head>\
```

```
    <title>$title</title>\
    <meta charset=\"utf-8\"/>\
   </head>\
  <body>\
  <h1>$title</h1>\n" if $. == 1;
  s/^(ARGUMENT|I{0,3}V?I{0,2})\.$/<h2>$1<\/h2>/; ❹
  if ($. == 5) { ❺
   s/^([A-Z].*)$/<p>$1<\/p>/;
  }
  if ($. == 9) { ❻
   s/^[ ]*(.*)/ <p>$1<br\/>/;
  }
  if (10..832) { ❼
  s/^([ ]{5,7}.*)/$1<br\/>/;
  }
  if (9..eof) { ❽
  s/^$/<br\/>/;
  }
  if ($. == 833) { ❾
   s/^(.*)$/$1<\/p>\n <\/body>\n<\/html>\n/;
  }
```

❶ 这一行是 shebang 命令，它提示 shell 你要运行的程序的位置。

❷ 如果当前行（`$.`）是第 1 行，则将整行（`$_`）赋值给变量 `$title`，同时使用 `chomp` 函数将字符串最后一个字符（换行符）剔除。

❸ 在文档顶部的第 1 行打印文档类型以及一些 HTML 标签，然后在几个地方重用变量 `$title` 的值。

❹ 用 `h2` 标签包括 ARGUMENT 标题和罗马数字。

❺ 用 `p` 标签包括介绍段落。

❻ 在诗文开头添加 `p` 起始标签，在行尾添加 `br` 标签。

❼ 在每行诗文结尾处添加一个 `br` 标签，最后一行除外。

❽ 在第 9 行后，每一个空行都用换行标签（`br`）代替。

❾ 在最后一行附加 `p`、`body` 以及 `html` 结束标签。

要运行该文件，只需输入：

```
perl html.pl rime.txt
```

还可以用 `>` 将输出内容重定向到文件中。在下一章也就是最后一章中，我们来总结本书讲解的正则表达式知识。

9.5　本章所学

- 如何在命令行中使用 sed
- 如何使用 sed 在前方添加（插入）、替换以及在尾部附加文本（及标签）
- 如何使用 Perl 做同样的事情

9.6　相关资源

- Stuart Rackham 的 AsciiDoc（http://www.methods.co.nz/asciidoc/）是一种文本格式，可以用 Python 预处理程序将其转换为 HTML、PDF、ePUB、DocBook 以及技术文档。该文本文件的语法与 Wiki 或 Markdown 相似，而且比手工编写 HTML 或 XML 标签要快很多。
- 下划线只适用于 XML 的标签名，而不适用于 HTML。另外，XML 标签名可用的字符范围要比用 ASCII 字符集表示的范围大很多。有关 XML 标签名可用字符的信息，请参见 http://www.w3.org/TR/REC-xml/#sec-common-syn。
- 如果 `perldoc` 命令不起作用，你还有其他选择。首先，可以直接到 http://perldoc.perl.org 在线阅读有关 Perl 的内容。（例如，要了解更多有关 `$.` 的内容，可以访问 http://perldoc.perl.org/perlvar.html#Variables-related-to-filehandles）。如果你运行的是 Mac，则可以试试 `perldoc5.12`。如果你是从 ActiveState 安装的 Perl，可以在 /usr/local/ActivePerl-5.XX/bin 中找到它。如果你是从源代码编译和构建的程序，则 `perl` 和 `perldoc` 都是安装在 /usr/local/bin 中的。可以将 /usr/local/bin 添加到你的路径变量中以方便运行 `perl` 和 `perldoc`。有关设置路径变量的信息，请参见 http://java.com/en/download/help/path.xml。

第10章

初级班毕业了

“Unix不会阻止用户干蠢事，因为那样也会妨碍用户做聪明的事儿。”

——Doug Gwyn

恭喜你坚持到正则表达式初级班毕业。你不再是个初学者了，你已经接触了最常用的正则表达式语法。作为程序员，正则表达式会给你的工作提供更多的改善机会。

学习正则表达式为我节省了不少的时间。让我举个例子吧。

我在工作中经常要使用 XSLT，经常要分析 XML 文件中的标签。

在上一章中我展示过一部分，但现在这里有个更长的单行命令，它从 lorem.dita 文件中提取一列标签名并转换为简单的 XSLT 样式表：

```
grep -Eo '<[_a-zA-Z][^>]*>' lorem.dita | sort | uniq | sed '1 i\
<xml:stylsheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">\

; s/^</\
<xsl:template match="/;s/ id=\".*\"//;s/>$/">\
 <xsl:apply-templates\/>\
<\/xsl:template>/;$ a\
\
</xsl:stylesheet>\
'
```

我知道这个脚本看起来很复杂，但你若长期使用这些东西之后，会培养出新的思维方式。我不打算解释这个脚本执行的操作了，因为我确信你可以自己弄明白。

该脚本的输出如下：

```
<xml:stylsheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="body">
 <xsl:apply-templates/>
</xsl:template>

<xsl:template match="li">
 <xsl:apply-templates/>
</xsl:template>

<xsl:template match="p">
 <xsl:apply-templates/>
</xsl:template>

<xsl:template match="title">
 <xsl:apply-templates/>
</xsl:template>

<xsl:template match="topic">
 <xsl:apply-templates/>
</xsl:template>

<xsl:template match="ul">
 <xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```

这只是个开头。当然，要让这个简单的样式表变得有用还需要做很多编辑工作，但这样可以让你少键入很多东西。

我承认如果在文件中用以下 sed 命令则会更简单。事实上，我就这样做了。你可以在示例代码库中找到 xslt.sed 文件。下面是该文件的内容：

```
#!/usr/bin/sed

1 i\
<xml:stylsheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">\

s/^</\
<xsl:template match="/;s/ id=\".*\"//;s/>$/">\
 <xsl:apply-templates\/>\
<\/xsl:template>/;$ a\
\
</xsl:stylesheet>\
```

然后这样运行它：

```
grep -Eo '<[_a-zA-Z][^>]*>' lorem.dita | sort | uniq | sed -f xslt.sed
```

10.1 想上中级班

虽然你现在已经较好地掌握了正则表达式，但仍然有很多东西要学。关于深入学习正则表达式我有如下一些建议。

Jeffrey E. F. Friedl 的 *Mastering Regular Expressions, Third Edition* [1]（http://shop.oreilly.com/product/9780596528126.do）是正则表达式的权威指南，很多程序员都以此为参考。这本书内容全面而且写得很好，如果你要用正则表达式做重要的工作，那么就该将这本书放在你的书架上或者在你的电子阅读器中。

Jan Goyvaerts 和 Steven Levithan 合著的 *Regular Expressions Cookbook* [2]（http://shop.oreilly.com/product/0636920023630.do）也是一本好书，尤其适用于对比不同的实现程序，也是一本必备书。

Tony Stubblebine 的 *Regular Expression Pocket Reference: Regular Expressions for Perl, Ruby, PHP, Python, C, Java and .NET*（http://shop.oreilly.com/product/9780596514273.do）是一本 128 页的参考书，虽然已出版七年了，但仍然很受欢迎。

Andrew Watt 的 *Beginning Regular Expressions*（Wrox, 2005）[3] 备受推崇。另外，Bruce Barnett 的在线 sed 教程非常有用（请参见 http://www.grymoire.com/Unix/Sed.html）。他展示了 sed 很多不易理解的特性，本书未涉及这些特性。

10.2 工具、实现程序以及程序库

本书已经展示了很多工具、实现程序以及程序库。在此我们再回顾一下，并了解几个新的。

10.2.1 Perl

Perl 是一个流行的通用编程语言。在通过正则表达式处理文本时，很多人更喜欢使用 Perl 语言，而不是其他语言。你可能已经安装了 Perl，但我们还是得提一下，要获取有关如何在系统中安装 Perl 的信息，请访问 http://www.perl.org/get.html。请到 http://perldoc.perl.org/perlre.html 阅读有关 Perl 的正则表达式的内容。当然，不要误解，我的意思并非是让大家都来学习 Perl。在其他很多语言中同样可以用正则表达式出色地完成任务，但有必要在你的工具箱中保留 Perl。要深入学习 Perl，你可以买一本 Randal Schwartz、brian d foy 和 Tom Phoenix 合著的 *Learning Perl, Sixth Edition* [4]，

注 1：中文版《精通正则表达式（第 3 版）》已由电子工业出版社出版（2007）。——编者注
注 2：《正则表达式经典实例》已经由东南大学出版社出版（2010）。——编者注
注 3：中文版《正则表达式入门经典》已由清华大学出版社出版（2008）。——编者注
注 4：中文版《Perl 语言入门（第 6 版）》已由东南大学出版社出版（2012）。——编者注

该书也是由 O’Reilly 出版的。

10.2.2 PCRE

PCRE（Perl Compatible Regular ExpressionPerl，Perl 兼容正则表达式）是一个用 C 语言编写的（8 位和 16 位）的正则表达式库，参见 http://www.pcre.org。该程序库主要包含一些函数，它们可以在任何 C 语言框架以及任何可使用 C 程序库的其他语言中被调用。从名字可以看出，它兼容 Perl 5 正则表达式，而且包含其他正则表达式实现程序的一些特性。Notepad++ 编辑器就使用了 PCRE 库。

pcregrep 是一个类似 grep 的 8 位工具程序，它可以让你在命令行中使用 PCRE 的特性。我们在第 3 章中使用过这个工具。关于下载信息请参见 http://www.pcre.org。可通过 Macports（http://www.macports.org）获取 Mac 版的 pcregrep，运行命令 `sudo port install pcre` 即可（必须预先安装 Xcode；见 https://developer.apple.com/technologies/tools/，需要注册）。要在 Windows 平台（二进制文件）上安装 pcregrep，请参考 http://gnuwin32.sourceforge.net/packages/pcre.htm。

10.2.3 Ruby（Oniguruma）

Oniguruma 是 Ruby 1.9 内置的正则表达式库（参见 http://oniguruma.rubyforge.org/）。它是用 C 语言编写的，专门支持 Ruby。可以用 Rubular 测试 Ruby 正则表达式，这个在线应用程序支持版本 1.8.7 和版本 1.9.2（见 http://www.rubular.com 和图 10-1）。顺便说一句，TextMate 就使用了 Oniguruma 库。

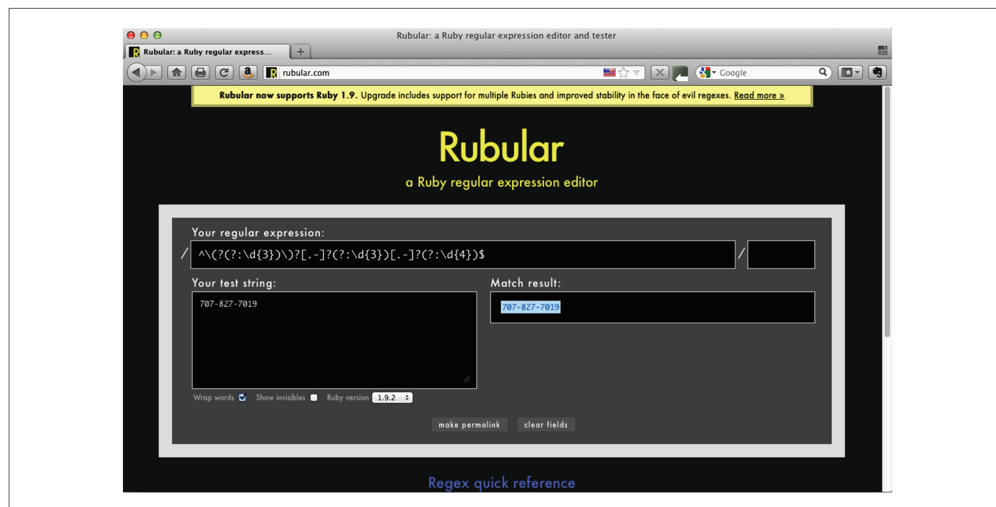


图 10-1：用 Rubular 匹配电话号码的正则表达式

10.2.4 Python

Python 是一种支持正则表达式的通用编程语言（参见 http://www.python.org），由 Guido van Rossum 于 1991 年开发。可以在以下链接阅读有关 Python 3 正则表达式的内容：http://docs.python.org/py3k/library/re.html?highlight=regular%20expressions。

10.2.5 RE2

RE2 是个非回溯的 C++ 正则表达式库（http://code.google.com/p/re2）。虽然 RE2 的速度很快，但是它不做回溯操作和后向引用。它是以 Perl 的 CPAN 包的形式来使用的，后向引用要依赖 Perl 的本地程序库来完成。关于 API 的内容，请参见 http://code.google.com/p/re2/wiki/CplusplusAPI。http://swtch.com/~rsc/regexp/regexp3.html 上这个题为“Regular Expression Matching in the Wild”的文章非常有趣，大家可以读读。

10.3 匹配北美电话号码

还记得第 1 章中匹配北美电话号码的例子吗？跟那时相比，你已经有了很明显的提高。

下面是一个更为可靠的匹配电话号码的正则表达式。这个正则表达式是由 Goyvaerts 和 Levithan 合著的 *Regular Expressions Cookbook*，*First Edition* 一书第 235 页的例子改编而来。

```
^\(?(?:\d{3})\)?[-.]?(?:\d{3})[-.]?(?:\d{4})$
```

请用手头的工具试一下这个表达式（图 10-2 显示的是在 Reggy 中输入该表达式的情况）。现在你应该能轻而易举地分析清楚这个表达式了。真为你骄傲！不过，还是让我们一块再分析一次吧。

- `^` 是判定一行或者主题词开头的零宽度断言。
- `\(?` 是一个字面左括号，但它是可选的（`?`）。
- `(?:\d{3})` 是一个匹配连续三位数字的非捕获分组。
- `\)?` 是可选的右括号。
- `[-.]?` 允许有可选的连字符或者句点（点号）。
- `(?:\d{3})` 是另一个匹配连续三位数字的非捕获分组。
- `[-.]?` 再次允许有可选的连字符或者句号（点号）。
- `(?:\d{4})` 是匹配连续四位数字的非捕获分组。
- `$` 匹配一行或主题词的结尾。

这个表达式还可以再改进，这项任务就留给你了，因为你已经可以自己做了。

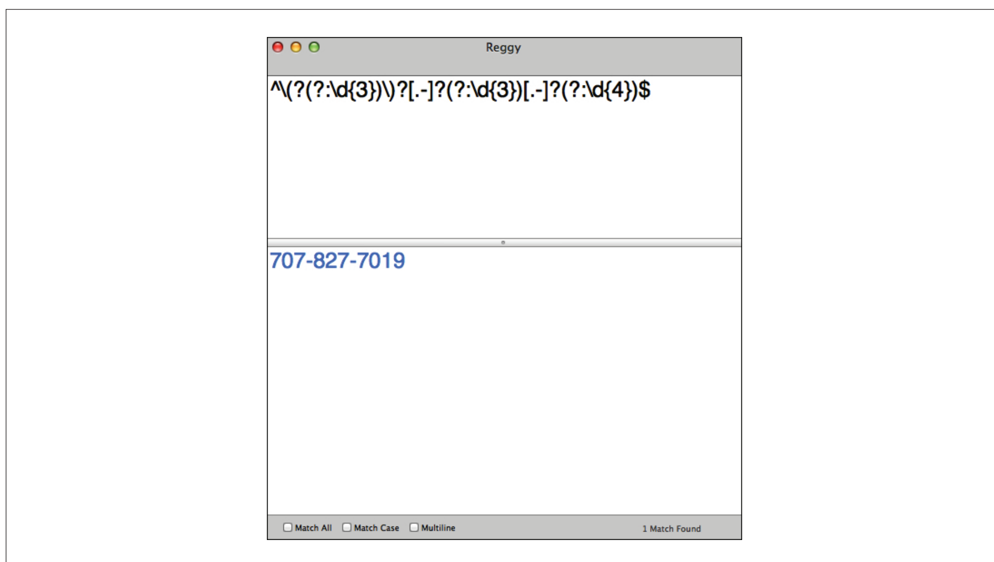


图 10-2：在 Reggy 中匹配电话号码的正则表达式

10.4　匹配电子邮件地址

最后，再给你一个匹配电子邮件地址的正则表达式：

```
^([\w-.!#$%&'*+-/=?^_`{|}~]+)@((?:\w+\.)+)(?:[a-zA-Z]{2,4})$
```

该表达式是由 Grant Skinner 用 RegExr 提供的表达式改编而来。我想让你尽力尝试从正则表达式的角度解释每一个字符的意思，然后看看有什么可以改进的地方。我确定你可以做到。

感谢你能拿出时间学习本书，很高兴和你共度这段时光。你现在应该已经很好地掌握了正则表达式的基本概念。你不再是新手了。希望你已经可以与正则表达式互动了，并且在学习过程中着实掌握了有用的东西。

10.5　本章所学

- 如何从文件中提取 XML 元素列表并将其转换为 XSLT 样式表
- 从哪里可以找到其他学习正则表达式的资源
- 值得推荐的正则表达式工具、实现程序以及程序库
- 更为可靠的匹配北美电话号码的模式

附录

正则表达式参考

本附录是正则表达式的参考资料。

QED中的正则表达式

QED（Quick Editor）最初是为运行在 Scientific Data Systems 公司生产的 SDS 940 上的 Berkeley Timesharing System 而编写的。QED 是 Ken Thompson 对 MIT 的 Compatible Time-Sharing System 上运行的一个编辑器重写后的版本，它是计算领域中最早的正则表达式实现程序之一。表 A-1 是从 1970 年贝尔实验室备忘录中选取的，它列出了 QED 中的正则表达式特性。直到 40 多年后的今天，这种语法在很大程度上仍在使用，这着实令人震惊。

表A-1：QED正则表达式

特　性	描　述
字面值	(a) 正则表达式，匹配该字符本身
^	(b) 正则表达式，匹配一行起始处的空字符
$	(c) 正则表达式，匹配字符换行符之前的空字符（通常在一行的结尾处）
.	(d) 正则表达式，匹配除换行符之外的任意字符
[<string>]	(e) 正则表达式，仅匹配字符串中的任意字符
[^<string>]	(f) 正则表达式，匹配除换行符和字符串中的字符之外的任意字符
*	(g) 正则表达式（由一个正则表达式后跟 * 构成），其匹配的是该正则表达式匹配的文本接连出现任意次（包括零次）所形成的内容
	(h) 正则表达式（由相邻的两个正则表达式构成），匹配的是两个正则表达式所匹配的文本毗邻出现的情况

（续）

<table>
<tr><th>特　　性</th><th>描　　述</th></tr>
<tr><td>|</td><td>(i) 正则表达式（由两个被 | 分隔的正则表达式构成），匹配的是两个正则表达式中的任何一个所匹配的文本</td></tr>
<tr><td>()</td><td>(j) 正则表达式（由正则表达式加括号组成），它与原表达式匹配的文本相同。括号是用来改变 (g)、(h) 以及 (i) 中隐含的求值顺序：a(b|c)d 会匹配 abd 或 acd，而 ab|cd 匹配的则是 ab 或 cd</td></tr>
<tr><td>{}</td><td>(k) 如果是一个正则表达式，则 {<regexp>}x 也是一个正则表达式，其中 x 是任意字符。该正则表达式与匹配相同的内容；它在替换命令中会有某些副作用。（替换命令的构成形式是 (.,.)S/<regexp>/<string>/（参见实验室备忘录的第 13 页），与如今它在 sed 和 Perl 等程序中的用法相似。）</td></tr>
<tr><td>\E</td><td>(l) 如果是由 E 命令命名的正则表达式名，则 \E 是一个正则表达式，它与 E 命令中指定的正则表达式匹配的内容相同。更多讨论见后面的 E 命令。”（\E 命令允 许对则表达式命名并根据名字来重复使用该表达式。）</td></tr>
<tr><td></td><td>(m) 单独的空正则表达式与前一个正则表达式等价。初始情况下未定义空正则表达式；在错误的正则表达式之后以及使用 E 命令后它也是未定义的</td></tr>
<tr><td></td><td>(n) 其他情形都不是正则表达式</td></tr>
<tr><td></td><td>(o) 正则表达式不匹配多行文本</td></tr>
</table>

元字符

正则表达式中使用的 14 个元字符分别有相应的特殊含义，如表 A-2 所示。如果你想将这些字符作为字面值使用，必须在该字符前加一个反斜线将其转义。例如，你可以像这样 \$ 将美元符转义，或像这样 \\ 将反斜线转义。

表A-2：正则表达式中的元字符

<table>
<tr><th>元字符</th><th>名　　称</th><th>代码点</th><th>作　　用</th></tr>
<tr><td>.</td><td>句点</td><td>U+002E</td><td>匹配任意字符</td></tr>
<tr><td>\</td><td>反斜线</td><td>U+005C</td><td>对字符转义</td></tr>
<tr><td>|</td><td>竖线符</td><td>U+007C</td><td>选择操作（或）</td></tr>
<tr><td>^</td><td>脱字符</td><td>U+005E</td><td>行起始锚位符</td></tr>
<tr><td>$</td><td>美元符</td><td>U+0024</td><td>行结束锚位符</td></tr>
<tr><td>?</td><td>问号</td><td>U+003F</td><td>匹配零次或一次的量词</td></tr>
<tr><td>*</td><td>星号</td><td>U+002A</td><td>匹配零次或多次的量词</td></tr>
<tr><td>+</td><td>加号</td><td>U+002B</td><td>匹配一次或多次的量词</td></tr>
<tr><td>[</td><td>左方括号</td><td>U+005B</td><td>字符组起始</td></tr>
<tr><td>]</td><td>右方括号</td><td>U+005D</td><td>字符组结束</td></tr>
<tr><td>{</td><td>左花括号</td><td>U+007B</td><td>量词或代码块起始</td></tr>
<tr><td>}</td><td>右花括号</td><td>U+007D</td><td>量词或代码块结束</td></tr>
<tr><td>(</td><td>左括号</td><td>U+0028</td><td>分组起始</td></tr>
<tr><td>)</td><td>右括号</td><td>U+0029</td><td>分组结束</td></tr>
</table>

字符简写式

表 A-3 列出了正则表达式中使用的字符简写式。

表A-3：字符简写式

字符简写式	描　　述	字符简写式	描　　述
`\a`	报警符	`\oxxx`	字符的八进制值
`\b`	单词边界	`\s`	空白符
`[\b]`	退格字符	`\S`	非空白符
`\B`	非单词边界	`\t`	水平制表符
`\cx`	控制字符	`\v`	垂直制表符
`\d`	数字字符	`\w`	单词字符
`\D`	非数字字符	`\W`	非单词字符
`\dxxx`	字符的十进制值	`\0`	空字符
`\f`	换页符	`\x xx`	字符的十六进制值
`\r`	回车符	`\uxxxx`	字符的 Unicode 值
`\n`	换行符		

空白符

表 A-4 列出了各种空白符的简写式。

表A-4：空白符

字符简写式	描　　述	字符简写式	描　　述
`\f`	换页符	`\r`	回车符
`\h`	水平空白符	`\t`	水平制表符
`\H`	非水平空白符	`\v`	垂直制表符
`\n`	换行符	`\V`	非垂直制表符

Unicode空白字符

表 A-5 列出的是 Unicode 中的各种空白字符。

表A-5：Unicode中的各种空白字符

简写或别名	名　　称	代码点	正则表达式
HT	水平制表符	U+0009	\u0009 或 `\t`
LF	换行符	U+000A	\u000A 或 `\n`
VT	垂直制表符	U+000B	\u000B 或 `\v`
FF	换页符	U+000C	\u000C 或 `\f`

（续）

简写或别名	名　　称	代码点	正则表达式
CR	回车符	U+000D	\u000d 或 \r
SP	空格符	U+0020	\u0020 或 \s*
NEL	下一行	U+0085	\u0085
NBSP	非间断空格	U+00A0	\u00A0
—	欧甘空格	U+1680	\u1680
MVS	蒙古文母音分隔符	U+180E	\u180E
BOM	字节顺序标记	U+FEFF	\ufeff
NQSP	半身空铅	U+2000	\u2000
MQSP，Mutton Quad	全身空铅	U+2001	\u2001
ENSP，Nut	单倍间距	U+2002	\u2002
EMSP，Mutton	双倍间距	U+2003	\u2003
3MSP，Thick space	三分之一全身间距	U+2004	\u2004
4MSP，Mid space	四分之一全身间距	U+2005	\u2005
6/MSP	六分之一全身间距	U+2006	\u2006
FSP	数字间隔	U+2007	\u2007
PSP	标点间隔	U+2008	\u2008
THSP	窄空格	U+2009	\u2009
HSP	细空格	U+200A	\u200A
ZWSP	零宽度空格	U+200B	\u200B
LSEP	行列分隔符	U+2028	\u2028
PSEP	段落分隔符	U+2029	\u2029
NNBSP	窄式不换行空格	U+202F	\u202F
MMSP	中等数学间隔	U+205F	\u205f
IDSP	表意字符间隔	U+3000	\u3000

* 也会匹配其他空白符。

控制字符

表 A-6 所示的是在正则表达式中匹配控制字符的一种方式。

表A-6：匹配控制字符

控制字符	Unicode值	简　　写	名　　称
c@*	U+0000	NUL	空字符
\cA	U+0001	SOH	标题起始
\cB	U+0002	STX	文本起始
\cC	U+0003	ETX	文本结束
\cD	U+0004	EOT	传输结束
\cE	U+0005	ENQ	询问

（续）

控制字符	Unicode值	简　写	名　称
\cF	U+0006	ACK	确认
\cG	U+0007	BEL	报警符
\cH	U+0008	BS	退格符
\cI	U+0009	HT	水平制表符
\cJ	U+000A	LF	换行符
\cK	U+000B	VT	垂直制表符
\cL	U+000C	FF	换页
\cM	U+000D	CR	回车符
\cN	U+000E	SO	移出
\cO	U+000F	SI	移入
\cP	U+0010	DLE	数据链转义
\cQ	U+0011	DC1	设备控制符一
\cR	U+0012	DC2	设备控制符二
\cS	U+0013	DC3	设备控制符三
\cT	U+0014	DC4	设备控制符四
\cU	U+0015	NAK	否定性确认
\cV	U+0016	SYN	同步空闲
\cW	U+0017	ETB	传输块结尾
\cX	U+0018	CAN	取消
\cY	U+0019	EM	介质结尾
\cZ	U+001A	SUB	替换
\c[	U+001B	ESC	转义
\c\	U+001C	FS	信息分隔符四
\c]	U+001D	GS	信息分隔符三
\c^	U+001E	RS	信息分隔符二
\c_	U+001F	US	信息分隔符一

* 可以使用大写或者小写形式。例如，\cA 与 \ca 是等价的；但是 Java 实现版本要求大写。

字符属性

表 A-7 列出了以 \p{ 属性 } 或 \P{ 属性 } 形式使用的字符属性名称。

表A-7：字符属性*

属　性	描　述	属　性	描　述	属　性	描　述
C	其他字符	Co	专用字符	Lm	修饰字母
Cc	控制字符	Cs	替代字符	Lo	其他字母
Cf	格式字符	L	字母	Lt	标题字母
Cn	未分配字符	Ll	小写字母	Lu	大写字母

（续）

属　性	描　述	属　性	描　述	属　性	描　述
`L&`	Ll、Lu 或者 Lt	`P`	标点符号	`Sc`	货币符号
`M`	标记符号	`Pc`	连接标点	`Sk`	修饰符号
`Mc`	空格标记符	`Pd`	破折号	`Sm`	数学符号
`Me`	环绕标记符	`Pe`	结束标点符	`So`	其他符号
`Mn`	非空格标记符	`Pf`	最终标点符	`Z`	分隔符
`N`	数字	`Pi`	起始标点符	`Zl`	行分隔符
`Nd`	十进制数字	`Po`	其他标点符	`Zp`	段落分隔符
`Nl`	字母数字	`Ps`	开始标点符	`Zs`	空格分隔符
`No`	其他数字	`S`	符号		

* 见 http://www.pcre.org/pcre.txt 中的 pcresyntax(3)。

各种字符属性的脚本名称

表 A-8 列出了以 `\p{` 属性 `}` 或 `\P{` 属性 `}` 形式使用的语言脚本名称。

表A-8：脚本名称*

Arabic (Arab)	Glagolitic (Glag)	Lepcha (Lepc)	Samaritan (Samr)
Armenian (Armn)	Gothic (Goth)	Limbu (Limb)	Saurashtra (Saur)
Avestan (Avst)	Greek (Grek)	Linear B (Linb)	Shavian (Shaw)
Balinese (Bali)	Gujarati (Gujr)	Lisu (Lisu)	Sinhala (Sinh)
Bamum (Bamu)	Gurmukhi (Guru)	Lycian (Lyci)	Sundanese (Sund)
Bengali (Beng)	Han (Hani)	Lydian (Lydi)	Syloti Nagri (Sylo)
Bopomofo (Bopo)	Hangul (Hang)	Malayalam (Mlym)	Syriac (Syrc)
Braille (Brai)	Hanunoo (Hano)	Meetei Mayek (Mtei)	Tagalog (Tglg)
Buginese (Bugi)	Hebrew (Hebr)	Mongolian (Mong)	Tagbanwa (Tagb)
Buhid (Buhd)	Hiragana (Hira)	Myanmar (Mymr)	Tai Le (Tale)
Canadian Aboriginal (Cans)	Hrkt: Katakana or Hiragana)	New Tai Lue (Talu)	Tai Tham (Lana)
Carian (Cari)	Imperial Aramaic (Armi)	Nko (Nkoo)	Tai Viet (Tavt)
Cham (None)	Inherited (Zinh/Qaai)	Ogham (Ogam)	Tamil (Taml)
Cherokee (Cher)	Inscriptional Pahlavi (Phli)	Ol Chiki (Olck)	Telugu (Telu)
Common (Zyyy)	Inscriptional Parthian (Prti)	Old Italic (Ital)	Thaana (Thaa)
Coptic (Copt/Qaac)	Javanese (Java)	Old Persian (Xpeo)	Thai (None)
Cuneiform (Xsux)	Kaithi (Kthi)	Old South Arabian (Sarb)	Tibetan (Tibt)
Cypriot (Cprt)	Kannada (Knda)	Old Turkic (Orkh)	Tifinagh (Tfng)
Cyrillic (Cyrl)	Katakana (Kana)	Oriya (Orya)	Ugaritic (Ugar)
Deseret (Dsrt)	Kayah Li (Kali)	Osmanya (Osma)	Unknown (Zzzz)
Devanagari (Deva)	Kharoshthi (Khar)	Phags Pa (Phag)	Vai (Vaii)
Egyptian Hieroglyphs (Egyp)	Khmer (Khmr)	Phoenician (Phnx)	Yi (Yiii)
Ethiopic (Ethi)	Lao (Laoo)	Rejang (Rjng)	
Georgian (Geor)	Latin (Latn)	Runic (Runr)	

* 参见 http://www.pcre.org/pcre.txt 或 http://ruby.runpaint.org/regexps#properties 中的 pcresyntax(3)。

POSIX字符组

表 A-9 列出了各种 POSIX 字符组。

表A-9：POSIX字符组

字　符　组	描　　述
[[:alnum:]]	字母数字字符（字母和数字）
[[:alpha:]]	字母字符（字母）
[[:ascii:]]	ASCII 字符（总共 128 个）
[[:blank:]]	空白字符
[[:ctrl:]]	控制字符
[[:digit:]]	数字
[[:graph:]]	图形字符
[[:lower:]]	小写字母
[[:print:]]	可打印字符
[[:punct:]]	标点符号
[[:space:]]	空格字符
[[:upper:]]	大写字母
[[:word:]]	单词字符
[[:xdigit:]]	十六进制数字

选项与修饰符

表 A-10 和表 A-11 分别列出了正则表达式中的选项和修饰符。

表A-10：正则表达式中的选项

选　　项	描　　述	支持平台
(?d)	Unix 中的行	Java
(?i)	不区分大小写	PCRE、Perl、Java
(?J)	允许重复的名字	PCRE*
(?m)	多行	PCRE、Perl、Java
(?s)	单行（dotall）	PCRE、Perl、Java
(?u)	Unicode	Java
(?U)	默认最短匹配	PCRE
(?x)	忽略空格和注释	PCRE、Perl、Java
(?-…)	撤销设置或关闭选项	PCRE

* 参见 http://www.pcre.org/pcre.txt 中的“Named Subpatterns”（命名子模式）。

表A-11：Perl语言中的修饰符（标记符）*

修　饰　符	描　　述
a	匹配 \d、\s、\w 以及处于 ASCII 范围内的 POSIX 字符
c	匹配失败后则停留在当前位置
d	使用默认的本地平台的规则
g	全局匹配
i	匹配时不区分大小写
l	使用当前位置的规则
m	多行字符串
p	保留匹配的字符串
s	将字符串看做一行内容
u	匹配时使用 Unicode 规则
x	忽略空格及注释

* 参见 http://perldoc.perl.org/perlre.html#Modifiers。

正则表达式与ASCII码表

表 A-12 是正则表达式与 ASCII 码表对照。

表A-12：ASCII码表

二进制	八进制	十进制	十六进制	字符	键盘	正则表达式	名　　称
00000000	0	0	0	NUL	^@	\c@	空字符
00000001	1	1	1	SOH	^A	\cA	标题开始
00000010	2	2	2	STX	^B	\cB	文本开始
00000011	3	3	3	ETX	^C	\cC	文本结束
00000100	4	4	4	EOT	^D	\cD	传输结束
00000101	5	5	5	ENQ	^E	\cE	询问
00000110	6	6	6	ACK	^F	\cF	确认
00000111	7	7	7	BEL	^G	\a，\cG	警报字符
00001000	10	8	8	BS	^H	[\b]，\cH	退格符
00001001	11	9	9	HT	^I	\t，\cI	水平制表符
00001010	12	10	0A	LF	^J	\n，\cJ	换行符
00001011	13	11	0B	VT	^K	\v，\cK	垂直制表符
00001100	14	12	0C	FF	^L	\f，\cL	换页符
00001101	15	13	0D	CR	^M	\r，\cM	回车符
00001110	16	14	0E	SO	^N	\cN	移出
00001111	17	15	0F	SI	^O	\cO	移入
00010000	20	16	10	DLE	^P	\cP	数据链转义
00010001	21	17	11	DC1	^Q	\cQ	设备控制 1（XON）
00010010	22	18	12	DC2	^R	\cR	设备控制 2

（续）

二进制	八进制	十进制	十六进制	字符	键盘	正则表达式	名　称
00010011	23	19	13	DC3	^S	\cS	设备控制 3（XOFF）
00010100	24	20	14	DC4	^T	\cT	设备控制 4
00010101	25	21	15	NAK	^U	\cU	否定性确认
00010110	26	22	16	SYN	^V	\cV	同步空闲
00010111	27	23	17	ETB	^W	\cW	传输块结尾
00011000	30	24	18	CAN	^X	\cX	取消
00011001	31	25	19	EM	^Y	\cY	介质结尾
00011010	32	26	1A	SUB	^Z	\cZ	替换
00011011	33	27	1B	ESC	^[	\e，\c[	转义
00011100	34	28	1C	FS	^|	\c|	文件分隔符
00011101	35	29	1D	GS	^]	\c]	分组分隔符
00011110	36	30	1E	RS	^^	\c^	记录分隔符
00011111	37	31	1F	US	^_	\c_	单元分隔符
00100000	40	32	20	SP	SP	\s，[ ]	空格
00100001	41	33	21	!	!	!	感叹号
00100010	42	34	22	"	"	"	引号
00100011	43	35	23	#	#	#	井号
00100100	44	36	24	$	$	\$	美元符
00100101	45	37	25	%	%	%	百分号
00100110	46	38	26	&	&	&	和号
00100111	47	39	27	'	'	'	撇号
00101000	50	40	28	(	(	(，\(	左括号
00101001	51	41	29	)	)	)，\)	右括号
00101010	52	42	2A	*	*	*	星号
00101011	53	43	2B	+	+	+	加号
00101100	54	44	2C	,	,	,	逗号
00101101	55	45	2D	-	-	-	连字符
00101110	56	46	2E	.	.	\.，[.]	句号
00101111	57	47	2F	/	/	/	斜线
00110000	60	48	30	0	0	\d，[0]	数字 0
00110001	61	49	31	1	1	\d，[1]	数字 1
00110010	62	50	32	2	2	\d，[2]	数字 2
00110011	63	51	33	3	3	\d，[3]	数字 3
00110100	64	52	34	4	4	\d，[4]	数字 4
00110101	65	53	35	5	5	\d，[5]	数字 5
00110110	66	54	36	6	6	\d，[6]	数字 6
00110111	67	55	37	7	7	\d，[7]	数字 7
00111000	70	56	38	8	8	\d，[8]	数字 8

（续）

二进制	八进制	十进制	十六进制	字符	键盘	正则表达式	名　　称
00111001	71	57	39	9	9	\d，[9]	数字 9
00111010	72	58	3A	:	:	:	冒号
00111011	73	59	3B	;	;	;	分号
00111100	74	60	3C	<	<	<	小于号
00111101	75	61	3D	=	=	=	等于号
00111110	76	62	3E	>	>	>	大于号
00111111	77	63	3F	?	?	?	问号
01000000	100	64	40	@	@	@	at
01000001	101	65	41	A	A	\w，[A]	拉丁大写字母 A
01000010	102	66	42	B	B	\w，[B]	拉丁大写字母 B
01000011	103	67	43	C	C	\w，[C]	拉丁大写字母 C
01000100	104	68	44	D	D	\w，[D]	拉丁大写字母 D
01000101	105	69	45	E	E	\w，[E]	拉丁大写字母 E
01000110	106	70	46	F	F	\w，[F]	拉丁大写字母 F
01000111	107	71	47	G	G	\w，[G]	拉丁大写字母 G
01001000	110	72	48	H	H	\w，[H]	拉丁大写字母 H
01001001	111	73	49	I	I	\w，[I]	拉丁大写字母 I
01001010	112	74	4A	J	J	\w，[J]	拉丁大写字母 J
01001011	113	75	4B	K	K	\w，[K]	拉丁大写字母 K
01001100	114	76	4C	L	L	\w，[L]	拉丁大写字母 L
01001101	115	77	4D	M	M	\w，[M]	拉丁大写字母 M
01001110	116	78	4E	N	N	\w，[N]	拉丁大写字母 N
01001111	117	79	4F	O	O	\w，[O]	拉丁大写字母 O
01010000	120	80	50	P	P	\w，[P]	拉丁大写字母 P
01010001	121	81	51	Q	Q	\w，[Q]	拉丁大写字母 Q
01010010	122	82	52	R	R	\w，[R]	拉丁大写字母 R
01010011	123	83	53	S	S	\w，[S]	拉丁大写字母 S
01010100	124	84	54	T	T	\w，[T]	拉丁大写字母 T
01010101	125	85	55	U	U	\w，[U]	拉丁大写字母 U
01010110	126	86	56	V	V	\w，[V]	拉丁大写字母 V
01010111	127	87	57	W	W	\w，[W]	拉丁大写字母 W
01011000	130	88	58	X	X	\w，[X]	拉丁大写字母 X
01011001	131	89	59	Y	Y	\w，[Y]	拉丁大写字母 Y
01011010	132	90	5A	Z	Z	\w，[Z]	拉丁大写字母 Z
01011011	133	91	5B	[	[	\[	左方括号
01011100	134	92	5C	\	\	\	反斜线
01011101	135	93	5D	]	]	\]	右方括号
01011110	136	94	5E	^	^	^，[^]	抑扬符号

（续）

二进制	八进制	十进制	十六进制	字符	键盘	正则表达式	名　称
01011111	137	95	5F	_	_	_, [_]	下划线
00100000	140	96	60	`	`	\\`	重音符
01100001	141	97	61	a	a	\w, [a]	拉丁小写字母 A
01100010	142	98	62	b	b	\w, [b]	拉丁小写字母 B
01100011	143	99	63	c	c	\w, [c]	拉丁小写字母 C
01100100	144	100	64	d	d	\w, [d]	拉丁小写字母 D
01100101	145	101	65	e	e	\w, [e]	拉丁小写字母 E
01100110	146	102	66	f	f	\w, [f]	拉丁小写字母 F
01100111	147	103	67	g	g	\w, [g]	拉丁小写字母 G
01101000	150	104	68	h	h	\w, [h]	拉丁小写字母 H
01101001	151	105	69	i	i	\w, [i]	拉丁小写字母 I
01101010	152	106	6A	j	j	\w, [j]	拉丁小写字母 J
01101011	153	107	6B	k	k	\w, [k]	拉丁小写字母 K
01101100	154	108	6C	l	l	\w, [l]	拉丁小写字母 L
01101101	155	109	6D	m	m	\w, [m]	拉丁小写字母 M
01101110	156	110	6E	n	n	\w, [n]	拉丁小写字母 N
01101111	157	111	6F	o	o	\w, [o]	拉丁小写字母 O
01110000	160	112	70	p	p	\w, [p]	拉丁小写字母 P
01110001	161	113	71	q	q	\w, [q]	拉丁小写字母 Q
01110010	162	114	72	r	r	\w, [r]	拉丁小写字母 R
01110011	163	115	73	s	s	\w, [s]	拉丁小写字母 S
01110100	164	116	74	t	t	\w, [t]	拉丁小写字母 T
01110101	165	117	75	u	u	\w, [u]	拉丁小写字母 U
01110110	166	118	76	v	v	\w, [v]	拉丁小写字母 V
01110111	167	119	77	w	w	\w, [w]	拉丁小写字母 W
01111000	170	120	78	x	x	\w, [x]	拉丁小写字母 X
01111001	171	121	79	y	y	\w, [y]	拉丁小写字母 Y
01111010	172	122	7A	z	z	\w, [z]	拉丁小写字母 Z
01111011	173	123	7B	{	{	{	左花括号
01111100	174	124	7C	|	|	|	竖线
01111101	175	125	7D	}	}	}	右花括号
01111110	176	126	7E	~	~	\~	波浪符
01111111	177	127	7F	DEL	^?	\c?	删除

相关资源

请读者参考 Ken Thompson 和 Dennis Ritchie 的 QED 备忘手册 http://cm.bell-labs.com/cm/cs/who/dmr/qedman.pdf。

术语表

ASCII

美国信息交换标准代码（American Standard Code for Information Interchange）。它是 1960 年制定的一种针对英文（拉丁）字符的编码方案，含有 128 个字符。另见 Unicode。

BRE

见基本正则表达式。

ed

实现正则表达式的一种 Unix 行编辑器，由 Ken Thompson 于 1971 年开发。它是 sed 和 vi 的前身。

ERE

见扩展正则表达式。

grep

使用正则表达式来查找字符串的 Unix 命令行工具。grep 是由 Ken Thompson 于 1973 年开发，据说是由 ed 编辑器命令 `g/re/p`（global/regular expression/print）发展而来。grep 后来被 egrep（或 grep -E）取代，后者拥有更多元字符，如 |、+、?、(和)，但并没有被淘汰。grep 使用基本正则表达式，grep -E（或 egrep）使用扩展正则表达式。fgrep（grep -F）使用文字字符串来查找文件，而像 $、* 和 | 这样的元字符没有特殊含义。见基本正则表达式、扩展的正则表达式。

Perl

一种通用程序设计语言，它由 Larry Wall 于 1987 年开发。Perl 由于其对正则表达式的强大支持以及文本处理能力而闻名。见 http://www.perl.org。

piece（片段）

在 POSIX.1 的术语中，为正则表达式的一部分，通常是连接起来的。参见 POSIX。

POSIX

Unix 可移植操作系统接口（Portable Operating System Interface for Unix）。由电子电气工程师协会（IEEE）制定的一系列有关 Unix 的标准。最近的正则表达式 POSIX 标准是 POSIX.1-2008（参见 http://standards.ieee.org/findstds/standard/1003.1-2008.html）。

sed

一种接受正则表达式并对文本进行转换的 Unix 流编辑器。20 世纪 70 年代初由 Lee McMahon 于贝尔实验室开发。sed 的一个示例：`sed -n 's/this/that/g\' file.ext > new.ext`。sed 的 -E 选项表示要使用扩展正则表达式。另见扩展正则表达式。

Unicode

Unicode 是一个将世界的各种书写系统字符进行编码的体系。Unicode 中的每个字符都有一个数值代码点。Unicode 支持的字符超过 10 万个。在正则表达式中，Unicode 字符可指定为 `\uxxxx` 或 `\x{xxxx}`，其中 *x* 表示一个范围为 0-9、A-F（或 a-f）的十六进制数，占用一位到四位。例如，`\u00E9` 表示字符 é，即小写拉丁字母 e 加一个重音符。另见 http://www.unicode.org。

vi
Bill Joy 于 1976 年开发的一款使用正则表达式的 Unix 编辑器。vim 编辑器是 vi 的改良版本，它最初由 Bram Moolenaar 开发（见 http://www.vim.org）。我日常工作中通常会用到六七种编辑器，但用得最多的还是 vim。事实上，如果我不幸遭遇海难流落到荒岛上，但还可以选择一个文本编辑器，我肯定会选择 vim，毫无疑问。

vim
见 vi。

八进制字符（octal character）
正则表达式中的字符可以使用八进制表示。在正则表达式中，八进制形式的字符通过以下方法指定：[\oxx]\o*xx*，其中 *x* 表示 1-9 范围内的数，占用一位或两位。例如，\o 表示字符 é，小写拉丁字母 e 外加一个重音符。

标记符（flag）
见*修饰符*。

捕获分组（capturing group）
见*分组*。

出现约束（occurrence constraint）
见*量词*。

代码点（code point）
见 Unicode。

断言（assertion）
参见*零宽度断言*。

分类表达式（bracketed expression）
方括号中的正则表达式，如 `[a-f]`，即为 a 到 f 范围内的小写字母。另见*字符组*。

分支（branch）
在 POSIX.1 的术语中，表示正则表达式中多个部分的连接。另见 POSIX。

分组（group）
分组将正则表达式原子组合放入一对括号 `()` 中。在 grep 或 sed 等应用程序中（不使用 -E），你必须在括号前加一个反斜线，就像 `\)` 和 `\(`。分组有捕获分组和非捕获分组两种。捕获分组将捕获的分组存储在内存中以便再次使用，而非捕获分组不会存储。原子分组不进行回溯。另见*原子分组*。

工作缓冲区（work buffer）
见*模式空间*。

反前瞻（negative lookahead）
见*前瞻*。

反后顾（negative lookbehind）
见*后顾*。

非捕获分组（non-capuring group）
在括号中不被捕获（即存储于内存以便将来使用）的分组。非捕获分组的语法是 `(?:pattern)`。另见*分组*。

后顾（lookbehind）
一种正则表达式，它仅在另一个指定的正则表达式在其之前时才会匹配。正后顾使用语法 `regex(?<=regex)`。反后顾表示在该表达式之前的内容*不是*在其之前的正则表达式时才匹配。使用语法 `regex(?<!regex)`。

后向引用（backreference）
对之前用括号捕获的正则表达式内容进行引用，其形式是 `\1`、`\2` 等。

环视（lookaround）
见*前瞻*、*后顾*。

回溯（backtracking）
逐个字符回退，逐个尝试从而找到成功的匹配。用于贪心式匹配，不能在懒惰式或者占有式匹配中使用。正则表达式处理器数千次尝试匹配耗费非常多的计算资源会导致*灾难性回溯*。避免灾难性回溯的一种方法就是使用原子分组。另见*原子分组*、*贪心式匹配*、*懒惰式匹配*、*占有式匹配*。

基本正则表达式（basic regular expression）
正则表达式的一种早期实现方式，较为落后，多数人认为它已经过时。一般简称为 BRE，BRE 要求对字符转义后才可作为元字符使用，例如括号（`\{` 和 `\}`）。另见*扩展正则表达式*。

可组合性（composability）
“一种模式语言（或实际的程序设计语言）提供一些原子对象和一些构造方法。这些构造方法可以用来将原子对象组合为复合对象，然后这些复合对象又可以进一步组合为复合对象。语言的可组合性就是指构造的多种方法能够统一适用于该语言的所有对象，无论是原子对象还是复合对象……

可组合性改善了易学性和易用性。可组合性还可能改善复杂度与功能之间的关系：对于给定的复杂度，语言的组合性越强，其功能就越强大。”引自 James Clark 的“The Design of RELAX NG”，参见 http://www.thaiopensource.com/relaxng/design.html#section:5。

扩展正则表达式（extended regular expression）
简称 ERE，在基本正则表达式 BRE 的基础上添加了额外的功能，例如，选择操作（\|）以及量词（如? 和 +），egrep（extened grep）支持扩展正则表达式，因此都能进行以上操作。IEEE POSIX 标准 1003.2-1992 对这些新特性进行了描述。grep 使用 -E 选项（就和使用 egrep 一样）就表示使用扩展正则表达式而不是基本正则表达式。另见选择操作、基本正则表达式、grep。

懒惰式匹配（lazy match）
在尝试寻找匹配时，懒惰式匹配一次只消耗目标字符串的一个字符。它不进行回溯。另见回溯、贪心式匹配、占有式匹配。

量词（quantifier）
设定在尝试匹配时正则表达式会出现的次数。一种形式是括号中包含一个整数或由逗号分隔的一对整数，如 {3} 表示表达式会出现 3 次（对于使用基本正则表达式的旧工具程序，必须将括号转义，就像 \{3\}）。

其他的量词有 ?（零次或一次）、+（一次或多次）、*（零次或多次）。量词也为限定符（bound）或修饰符（modifier）。量词自身是贪心式的。此外还有懒惰量词（如 {3}?）和占有量词（如 {3}+）。另见基本正则表达式、贪心式匹配、懒惰式匹配、占有式匹配。

零宽度断言（zero-width assertion）
在匹配过程中不会消耗字符的边界。如 ^ 和 $，分别匹配行起始和行结束。

锚位符 （anchor）
指定行或字符串中的位置。例如，脱字符（也叫音调符 ^）表示行或字符串的起始，而美元符（$）则表示行或字符串的结束。

模式空间（pattern space）
sed 程序通常逐行处理输入内容。在处理每一行时，该行的内容就被放入模式空间中，在该空间中就会应用模式。这也被称为工作缓冲区（work buffer）。见容纳空间、sed。

匹配（matching）
正则表达式会在文件中匹配指定的模式，然后根据应用程序要求触发结果。

前瞻（lookahead）
一种正则表达式，它仅在另一个指定的正则表达式尾随其后时才会匹配。正前瞻使用语法 `regex(?=regex)`。反前瞻表示尾随该表达式的内容不是尾随其后的正则表达式时才匹配。使用语法 `regex(?!regex)`。

取反操作（negation）
表示一个正则表达式不匹配给定的模式。

容纳缓冲区（hold buffer）
见容纳空间。

容纳空间（hold space）
sed 用于存储一行或多行以便进行进一步处理的空间。也称为容纳缓冲区。参见模式空间、sed。

十六进制（hexadecimal）
由数字 0-9 和字母 A-F 或 a-f 表示的基数为 16 的计数系统。例如，十进制数 15 在十六进制中表示为 F，而 16 在十六进制中表示为 10。

贪心式匹配（greedy match）
贪心式匹配会尽可能多地消耗目标字符串中的字符，然后在字符串中回溯来尝试寻找匹配。见回溯、懒惰式匹配和占有式匹配。

限定符（bound）
见量词（quantifier）。

修饰符（modifier）
放在匹配或替换模式之后的字符，它会对匹配过程进行修改。例如，i 修饰符会将匹配变为不区分大小写。修饰符也称为标记符。

选项（option）
允许你开启或关闭选项来修改匹配内容。例如，(?i) 选项表示匹配是不分大小写的。这与修饰符功能相似，但是所用的语法不同。另见修饰符。

选择操作 （alternation）
用竖线符（|）将几个正则表达式分隔，表示“或”操作。也就是说，由一个或多个 | 分隔的任

何一个正则表达式匹配即可。在某些应用程序中，就像使用基本正则表达式（BRE）的 grep 或 sed，则需要在 | 之前加一个反斜线符号 \|。另见基本正则表达式。

原子（atom ）
参见元字符。

原子分组（atomic group）
一种分组模式，当（?>…）中的正则表达式无法匹配时，就关闭回溯操作。另见回溯、分组。

元字符（metacharacter）
一种在正则表达式中有特殊含义的字符。这些字符是 .、\、\|、*、+、?、~、$、[、]、(、)、{、}。元字符也称为原子。

灾难性回溯（catastrophic backtracking）
参见回溯。

占有式匹配（possessive match）
在尝试寻找匹配时，占有式匹配一次直接消耗整个字符串。它不进行回溯。另见回溯、贪心式匹配、懒惰式匹配。

正前瞻（positive lookahead）
见前瞻。

正后顾（positive lookbehind）
见后顾。

正则表达式（regular expression）
一种经过特别编写的字符串，在应用或工具程序中使用时可以匹配其他的字符串或字符串集合。20 世纪 50 年代，数学家 Stephen Kleene（1909—1994）在 1952 年出版的 *Introduction to Metamathematics* 一书中，讲到有关形式语言理论的工作时首次描述了正则表达式。Ken Thompson 等人有关 QED 编辑器（在运行于 GE-635 上的通用电气分时系统中）的工作，以及 20 世纪 70 年代早期 AT&T 贝尔实验室的 Unix 操作系统中其他工具程序的开发推动了正则表达式在计算机领域的发展。

转义字符（character escape）
前面带有反斜线的字符。例如，\t（水平制表符）、\v（垂直制表符）和 \f（换页符）。

字符集（character set）
见字符组。

字符组（character class）
通常是指包括在方括号中的一组字符，例如 [a-bA-B0-9] 就是一个字符组，它表示 ASCII 或者基本拉丁（Low Basic Latin）字符集范围内的所有的大小写字符以及数字。

字符串字面值（string literal）
仅从文字字面上解释的字符串，如文字字符串“It is an ancient Marinere”，对比“[Ii]t[ ]is[ ].*nere.”这类字符串。

字面值（literal）
见字符串字面值。

索引

符号表

作者及封面简介

关于作者

Michael Fitzgerald 是程序员、顾问，为 O'Reilly 以及 John Wiley&Sons 编写过十余本技术图书，在 O'Reilly Network 也发表过很多文章。他曾是针对 XML 的 RELAX NG 模式语言委员会成员。

封面动物

本书封面上的动物是果蝠。

大蝙蝠亚目以及狐蝠科蝙蝠称为果蝠、狐蝠、原始果蝠或大蝙蝠。尽管有“大蝙蝠”的别名，但狐蝠科成员在体型上差异很大，最小的只有六厘米，也有重达两磅，展翼后五英尺的。

果蝠以果实或花蜜为食。有些果蝠会用牙齿咬开果皮吃掉果肉，有些则舔食碎果实的汁液。而许多以花蜜为食的果蝠，则是传授花粉和传播种子的高手。事实上，世界蝙蝠保护区（World Bat Sanctuary）新生雨林约 95% 归功于果蝠对种子的传播。蝙蝠和植物之间的这种关系就是一种互利共生关系，也称为蝙蝠媒（chiropterophily），属于不同物种个体为了彼此利益相互影响的一种方式。

果蝠更喜欢温暖的热带气候，因为在这种气候中水果和花朵容易找到，但世界各地都能见到果蝠。果蝠擅长飞行，但落地非常笨拙；它们落地过程中常常撞在树上，有时候为了停止飞行会尽力用爪子抓住树枝。这就让人们误以为它们眼盲，而事实上果蝠的视力是所有蝙蝠中最好的。许多蝙蝠依靠回声避开障碍物，果蝠凭借视力以及出色的嗅觉来定位食物和进行导航。

封面图片来自 Cassell 的 *Natrual History*。

学习正则表达式

你是程序员？正物色一本学习正则表达式的入门图书？恭喜，《学习正则表达式》非常适合你！本书提供大量经典简洁的示例，从零开始教你逐步掌握正则表达式。通过匹配特定单词、字符和模式，读者很快就可以自己动手使用正则表达式匹配、提取和转换文本。

正则表达式是程序员必备的强大工具，得到了各种Unix实用程序，以及Perl、Java、JavaScript、C#等编程语言的支持。读完本书，你会对正则表达式的常用语法了然于胸。掌握正则表达式是提升编程效率、节约时间的一大法宝。本书主要内容如下。

- 正则表达式的基本概念和原理。
- 在命令行工具和各种编程语言中使用正则表达式。
- 应用简单方法在文本中查找字符串，包括数值、字母、Unicode字符、字符串字面值。
- 零宽度断言和环视的用法。
- 如何使用分组、后向引用、字符组和修饰符。
- 使用正则表达式为纯文本添加HTML5标记。

Michael Fitzgerald

知名程序员、顾问、技术作家，为O’Reilly以及John Wiley & Sons编写过十余本技术图书，在O’Reilly Network上发表了大量文章。他曾是针对XML的RELAX NG模式语言委员会的成员。

封面设计：Karen Montgomery　张健

图灵社区：www.ituring.com.cn
新浪微博：@图灵教育 @图灵社区
反馈/投稿/推荐信箱：contact@turingbook.com
热线：(010)51095186转604

分类建议　程序设计

人民邮电出版社网址：www.ptpress.com.cn

O’Reilly Media, Inc.授权人民邮电出版社出版

此简体中文版仅限于中国大陆（不包含中国香港、澳门特别行政区和中国台湾地区）销售发行
This Authorized Edition for sale only in the territory of People's Republic of China (excluding Hong Kong, Macao and Taiwan)

O’REILLY®
oreilly.com.cn





ISBN 978-7-115-31149-8
定价：35.00元

欢迎加入

图灵社区

最前沿的IT类电子书发售平台

电子出版的时代已经来临。在许多出版界同行还在犹豫彷徨的时候，图灵社区已经采取实际行动拥抱这个出版业巨变。作为国内第一家发售电子图书的IT类出版商，图灵社区目前为读者提供两种DRM-free的阅读体验：在线阅读和PDF。

相比纸质书，电子书具有许多明显的优势。它不仅发布快，更新容易，而且尽可能采用了彩色图片（即使有的书纸质版是黑白印刷的）。读者还可以方便地进行搜索、剪贴、复制和打印。

图灵社区进一步把传统出版流程与电子书出版业务紧密结合，目前已实现作译者网上交稿、编辑网上审稿、按章发布的电子出版模式。这种新的出版模式，我们称之为"敏捷出版"，它可以让读者以较快的速度了解到国外最新技术图书的内容，弥补以往翻译版技术书"出版即过时"的缺憾。同时，敏捷出版使得作、译、编、读的交流更为方便，可以提前消灭书稿中的错误，最大程度地保证图书出版的质量。

现在购买电子书,读者将获赠书款20%的社区银子,可用于兑换纸质样书。

最方便的开放出版平台

图灵社区向读者开放在线写作功能，协助你实现自出版和开源出版的梦想。利用"合集"功能，你就能联合二三好友共同创作一部技术参考书，以免费或收费的形式提供给读者。（收费形式须经过图灵社区立项评审。）这极大地降低了出版的门槛。只要你有写作的意愿，图灵社区就能帮助你实现这个梦想。成熟的书稿，有机会入选出版计划，同时出版纸质书。

图灵社区引进出版的外文图书，都将在立项后马上在社区公布。如果你有意翻译哪本图书，欢迎你来社区申请。只要你通过试译的考验，即可签约成为图灵的译者。当然，要想成功地完成一本书的翻译工作，是需要有坚强的毅力的。

最直接的读者交流平台

在图灵社区，你可以十分方便地写作文章、提交勘误、发表评论，以各种方式与作译者、编辑人员和其他读者进行交流互动。提交勘误还能够获赠社区银子。

你可以积极参与社区经常开展的访谈、审读、评选等多种活动，赢取积分和银子，积累个人声望。

ituring.com.cn